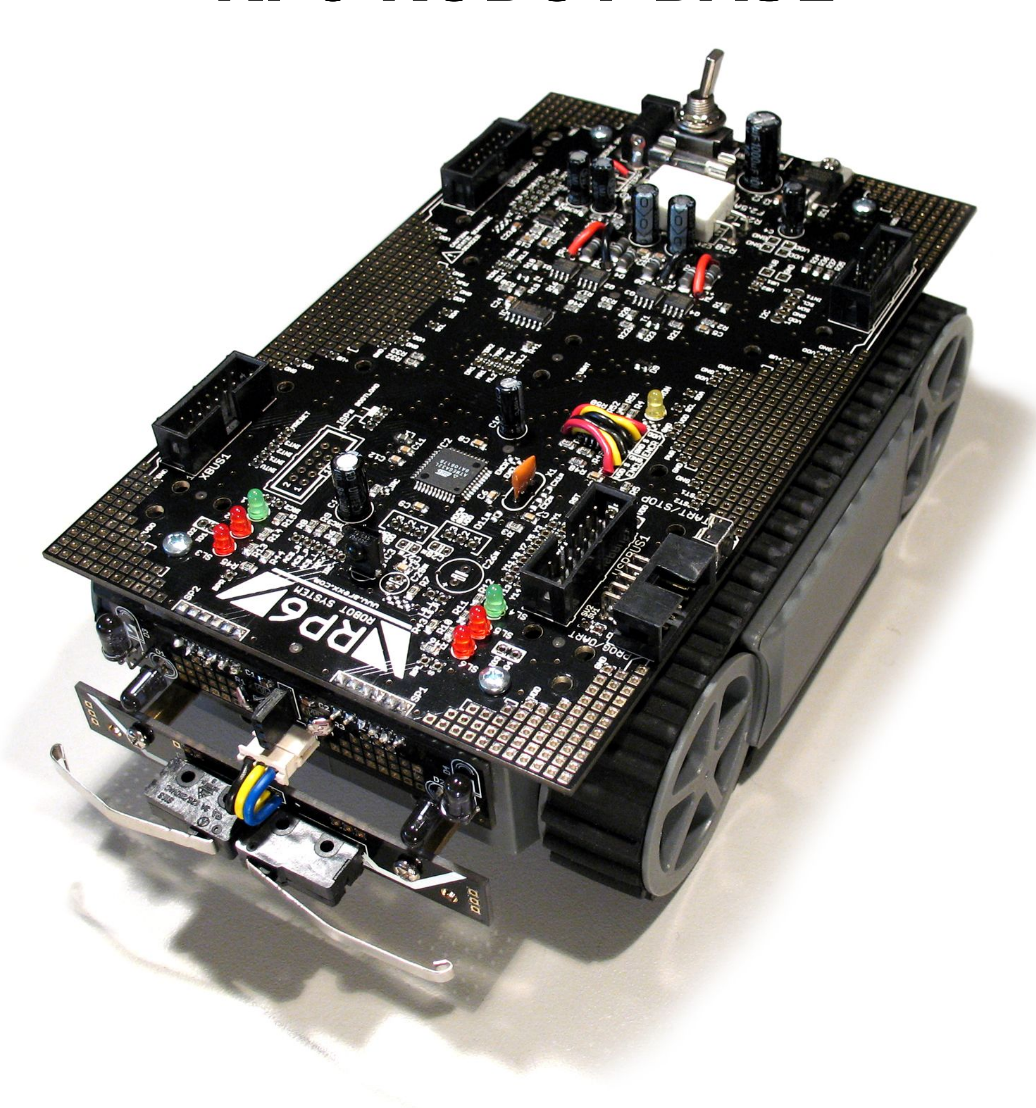
RP6 ROBOT SYSTEM

RP6 ROBOT BASE



RP6-BASE

www.arexx.com

RP6
Robot System

Bedienungsanleitung

- Deutsch (German) -

Version RP6-BASE-DE-20071219

WICHTIGE INFORMATION!
Bitte unbedingt lesen!

Bevor Sie den RP6 oder Zubehör in Betrieb nehmen, lesen Sie bitte diese Anleitung und ggf. die Anleitungen von Zubehörteilen vollständig durch! Sie erläutert Ihnen die korrekte Verwendung und weist auf mögliche Gefahren hin! Weiterhin enthält sie wichtige Informationen, die für viele Anwender keineswegs offensichtlich sein dürften.

Bei Nichtbeachtung dieser Anleitung erlischt jeglicher Garantieanspruch! Weiterhin übernimmt AREXX Engineering keinerlei Haftung für Schäden jeglicher Art, die aus Nichtbeachtung dieser Anleitung resultieren!

Bitte beachten Sie vor allem den Abschnitt "Sicherheitshinweise"!

Schließen Sie das USB Interface bitte erst an Ihren PC an, nachdem Sie das Kapitel 3 - „Inbetriebnahme" gelesen und die Software korrekt installiert haben!

Impressum

©2007 AREXX Engineering

Nervistraat 16
8013 RS Zwolle
The Netherlands

Tel.: +31 (0) 38 454 2028
Fax.: +31 (0) 38 452 4482

Änderungen an Produktspezifikationen und Lieferumfang vorbehalten.
Der Inhalt dieser Bedienungsanleitung kann jederzeit ohne vorherige Ankündigung geändert werden.
Neue Versionen dieser Anleitung erhalten Sie kostenlos auf http://www.arexx.com/

Wir sind nicht verantwortlich für den Inhalt von externen Webseiten, auf die in dieser Anleitung verlinkt wird!

Hinweise zur beschränkten Garantie und Haftung

Die Gewährleistung von AREXX Engineering beschränkt sich auf Austausch oder Reparatur des Roboters innerhalb der gesetzlichen Gewährleistungsfrist bei nachweislichen Produktionsfehlern, wie mechanischer Beschädigung und fehlender oder falscher Bestückung elektronischer Bauteile, ausgenommen aller über Steckverbinder/Sockel angeschlossenen Komponenten. Es besteht keine Haftbarkeit für Schäden, die unmittelbar durch, oder in Folge der Anwendung des Roboters entstehen. Unberührt davon bleiben Ansprüche, die auf unabdingbaren gesetzlichen Vorschriften zur Produkthaftung beruhen.

Sobald Sie irreversible Veränderungen (z.B. Anlöten von weiteren Bauteilen, Bohren von Löchern etc.) am Roboter vornehmen oder der Roboter Schaden infolge von Nichtbeachtung dieser Anleitung nimmt, erlischt jeglicher Garantieanspruch!

Es kann nicht garantiert werden, dass die mitgelieferte Software individuellen Ansprüchen genügt oder komplett unterbrechungs und fehlerfrei arbeiten kann.
Weiterhin ist die Software beliebig veränderbar und wird vom Anwender in das Gerät geladen. Daher trägt der Anwender das gesamte Risiko bezüglich der Qualität und der Leistungsfähigkeit des Gerätes inklusive aller Software.
AREXX Engineering garantiert die Funktionalität der mitgelieferten Applikationsbeispiele unter Einhaltung der in den technischen Daten spezifizierten Bedingungen. Sollte sich der Roboter oder die PC-Software darüber hinaus als fehlerhaft oder unzureichend erweisen, so übernimmt der Kunde alle entstehenden Kosten für Service, Reparatur oder Korrektur.
Bitte beachten Sie auch die entsprechenden Lizenzvereinbarungen auf der CD-ROM!

Symbole

Im Handbuch werden folgende Symbole verwendet:

Das "Achtung!" Symbol weist auf besonders wichtige Abschnitte hin, die sorgfältig beachtet werden müssen. Wenn Sie hier Fehler machen, könnte dies ggf. zur Zerstörung des Roboters oder seines Zubehörs führen und sogar Ihre eigene oder die Gesundheit anderer gefährden!

Das "Information" Symbol weist auf Abschnitte hin, die nützliche Tipps und Tricks oder Hintergrundinformationen enthalten. Hier ist es nicht immer essentiell alles zu verstehen, aber meist sehr nützlich.

Inhaltsverzeichnis

1. Einleitung

Der RP6 ist ein kostengünstiger autonomer mobiler Roboter, entwickelt um sowohl Anfängern, als auch fortgeschrittenen Elektronik und Software Entwicklern einen Einstieg in die faszinierende Welt der Robotik zu bieten.

Der Roboter wird komplett aufgebaut geliefert und ist damit ideal für alle, die sonst eher wenig mit Löten und Basteln zu tun haben und sich hauptsächlich auf die Software konzentrieren möchten. Allerdings bedeutet dies nicht, dass man keine eigenen Schaltungen und Erweiterungen auf dem Roboter aufbauen könnte! Ganz im Gegenteil: Der RP6 ist auf Erweiterung ausgelegt und kann als Ausgangspunkt für viele interessante Hardware Experimente verwendet werden!

Er tritt die Nachfolge des 2003 von der Conrad Electronic SE auf den Markt gebrachten und sehr erfolgreichen "C-Control Robby RP5" an (CCRP5, RP5 hieß dabei "Robot Project 5"), hat mit dessen Elektronik aber nur wenig gemeinsam. Der Mikrocontroller auf dem RP6 ist nicht mehr das C-Control 1 von Conrad Electronic und damit ist der Roboter auch nicht mehr direkt in Basic programmierbar. Stattdessen wird ein viel leistungsfähigerer ATMEGA32 von Atmel verwendet, der in C programmierbar ist. Es wird demnächst aber auch ein Erweiterungsmodul geben, mit dem neuere C-Control Varianten auf dem Roboter verwendet werden können. Damit kann der Roboter dann auch in der einfacheren Sprache Basic programmiert werden und gleichzeitig um viele weitere Schnittstellen und sehr viel zusätzlichen Speicher erweitert werden.

Weitere Neuerungen sind das direkt im Lieferumfang enthaltene USB Interface, ein sehr viel flexibleres Erweiterungssystem mit besseren Montagemöglichkeiten, stark verbesserte Drehgeber (150 fach höhere Auflösung im Vergleich zum Vorgänger), eine präzise Spannungsstabilisierung (beim Vorgänger war dazu noch ein Erweiterungsmodul notwendig), eine Stoßstange mit zwei Tastsensoren und vieles andere mehr. Das direkt im Lieferumfang enthaltene Experimentiermodul für eigene Schaltungen ist ebenfalls eine sehr nützliche Neuerung. Insgesamt ist das Preis/Leistungsverhältnis im Vergleich zum Vorgänger damit deutlich besser geworden.

Die Mechanik ist zwar grundsätzlich vom RP5 übernommen worden, wurde aber auf viel leiseren Betrieb optimiert und es wurden einige zusätzliche Bohrungen zur Montage von mechanischen Erweiterungen hinzugefügt.

Der RP6 ist vom Prozessor her kompatibel mit den Robotern ASURO und YETI, die beide den kleineren ATMEGA8 und dieselben Entwicklungswerkzeuge (WinAVR, avr-gcc) verwenden. ASURO und YETI kommen aber im Gegensatz zum RP6 als Bausätze und müssen vom Anwender aufgebaut werden. Der RP6 ist für etwas anspruchsvollere Anwender gedacht, die sehr gute Erweiterungsmöglichkeiten, bessere Mikrocontroller und mehr Sensoren benötigen.

Es sind bereits einige Erweiterungsmodule geplant oder sogar schon erhältlich, mit denen Sie die Fähigkeiten des Roboters ausbauen können. Dies ist u.a. die oben genannte C-Control Erweiterung, ein Erweiterungsmodul mit einem weiteren MEGA32 und natürlich das Lochraster Experimentierboard für eigene Schaltungen, welches auch separat erhältlich ist (man kann mehrere davon auf dem Roboter anbringen). Demnächst sind noch weitere interessante Module geplant und Sie können natürlich auch Ihre eigenen Erweiterungsmodule entwickeln!

Wir wünschen Ihnen nun viel Spaß und Erfolg mit dem RP6 Robot System!

1.1. Technischer Support

Bei Fragen oder Problemen erreichen Sie unser Support Team wie folgt über das Internet (Bevor Sie uns kontaktieren **lesen Sie aber bitte diese Bedienungsanleitung vollständig durch!** Erfahrungsgemäß erledigen sich viele Fragen so bereits von selbst! Bitte beachten Sie auch insbesondere Anhang A - Fehlersuche):

- über unser Forum: `http://www.arexx.com/forum/`

- per E-Mail: `info@arexx.nl`

Unsere Postanschrift finden Sie im Impressum dieses Handbuchs. Aktuellere Kontaktinformation, Softwareupdates und weitere Informationen gibt es auf unserer Homepage:

http://www.arexx.com/

und auf der Homepage des Roboters:

http://www.arexx.com/rp6

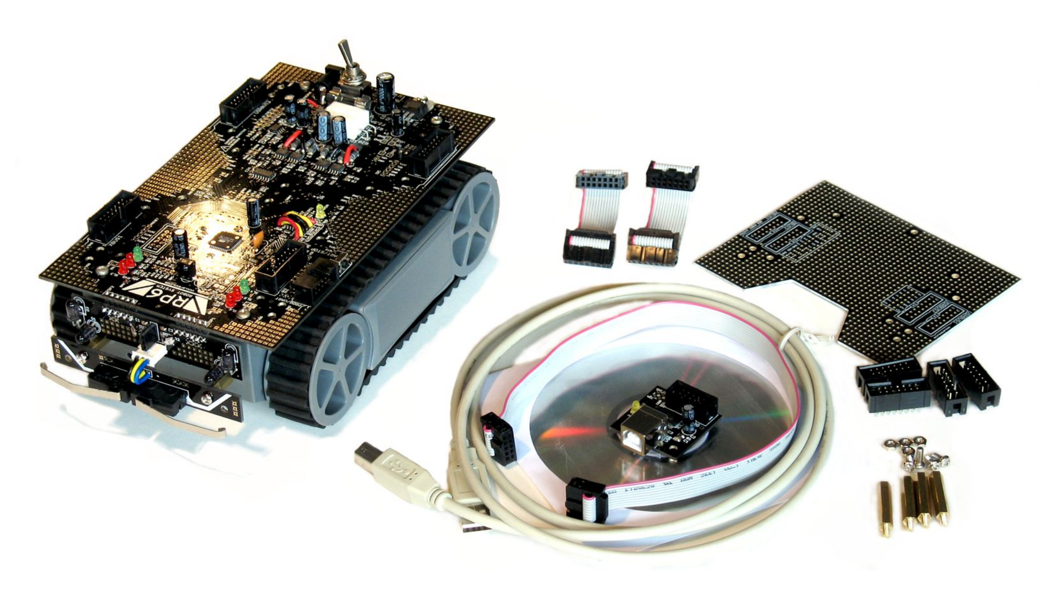
Auch dem Roboternetz, der größten deutschsprachigen Robotik Community, kann man auf jeden Fall mal einen Besuch abstatten:

http://www.roboternetz.de/

1.2. Lieferumfang

Folgende Dinge sollten Sie in Ihrem RP6 Karton vorfinden:

- **Fertig aufgebauter RP6 Roboter**
- RP6 USB Interface
- USB A->B Kabel
- 10 poliges Flachbandkabel
- RP6 CD-ROM
- Kurzanleitung
- **RP6 Experimentierplatine**
- 4 Stück 25mm M3 Distanzbolzen
- 4 Stück M3 Schrauben
- 4 Stück M3 Muttern
- 4 Stück 14 pol Wannenstecker
- 2 Stück 14 pol Flachbandkabel



1.3. Features und technische Daten

Dieser Abschnitt gibt einen Überblick darüber, was der Roboter zu bieten hat und dient gleichzeitig der Einführung einiger Begriffe und Bezeichnungen von Komponenten des Roboters, die vielen Anwendern noch nicht bekannt sein dürften. Vieles davon wird später im Handbuch noch detaillierter erläutert!

Features, Komponenten und technische Daten des RP6:

- **Leistungsfähiger Atmel ATMEGA32 8-Bit Mikrocontroller**
  - ◊ Geschwindigkeit 8 MIPS (=8 Millionen Instruktionen pro Sekunde) bei 8MHz Takt
  - ◊ Speicher: 32KB Flash ROM, 2KB SRAM, 1KB EEPROM
  - ◊ Frei in C programmierbar (mit WinAVR / avr-gcc)!
  - ◊ ... und vieles mehr! Weitere Details folgen in Kapitel 2.
- **Flexibles Erweiterungssystem, basierend auf dem I²C-Bus**
  - ◊ Nur zwei Signalleitungen erforderlich (TWI -> "Two Wire Interface")
  - ◊ Übertragungsgeschwindigkeit von bis zu 400kBit/s
  - ◊ Master->Slave basierend
  - ◊ Bis zu 127 Slaves können gleichzeitig am Bus angeschlossen werden
  - ◊ Weit verbreitetes Bussystem: Es sind sehr viele standard ICs, Sensoren und ähnliches von vielen verschiedenen Herstellern verfügbar, die meist direkt daran angeschlossen werden können.
- **Symmetrische Montagemöglichkeiten für Erweiterungsmodule an Front und Heck**
  - ◊ Es können theoretisch beliebig viele Erweiterungsmodule übereinander gestapelt werden, jedoch sind vom Energiebedarf/Gewicht her insgesamt nur etwa 6 bis 8 Module sinnvoll (--> jeweils 3 bis 4 Stück vorne und hinten).
  - ◊ 22 freie 3.2mm Montagelöcher sind auf dem Mainboard vorhanden, weitere 16 im Roboterchassis also insgesamt 38 Montagelöcher - und es gibt im Chassis noch sehr viel Platz für eigene Bohrungen!
- **Experimentierplatine direkt im Lieferumfang!** (s. Foto des Lieferumfangs)
- **USB PC Interface** für den Programupload vom PC auf den Mikrocontroller
  - ◊ Kabelgebunden für maximale Geschwindigkeit. Der Programmupload läuft normalerweise mit 500kBaud - der komplette freie Programmspeicher des Mikrocontrollers (30KB, 2KB sind für den sog. Bootloader reserviert) wird innerhalb von wenigen Sekunden beschrieben.
  - ◊ Das Interface kann zum Programmieren von allen für den RP6 erhältlichen Erweiterungsmodulen mit AVR Mikrocontroller verwendet werden.
  - ◊ Kann zur Kommunikation mit dem Roboter oder mit Erweiterungsmodulen verwendet werden - dadurch erleichtert sich die Fehlersuche in Programmen erheblich, da man Messwerte, Textnachrichten und andere Daten darüber an den PC senden kann.

◊ Der Treiber des Interfaces erzeugt eine virtuelle serielle Schnittstelle unter allen gängigen Betriebssystemen wie Windows 2K/XP und Linux, die mit fast allen Terminalprogrammen und eigener Software benutzbar ist.

◊ Für den komfortablen Programmupload wird die Software **RP6Loader** mitgeliefert. Sie bietet auch direkt ein kleines Terminal um mit dem Roboter über Textnachrichten zu kommunizieren und läuft unter Windows und Linux.

- **Leistungsfähiges Raupen-Antriebssystem** mit neuem Getriebe zur Minimierung der Geräuschentwicklung *(im Vergleich zum Vorgänger CCRP5...)*

◊ Zwei starke 7.2V DC-Motoren

◊ Maximale Geschwindigkeit 25cm/s (30cm/s ohne Software Begrenzung). Das hängt u.a. von Ladezustand/Qualität der Akkus und Gesamtgewicht ab!

◊ Selbstschmierende Sinterlager an allen vier 4mm Radachsen

◊ Zwei Gummi-Raupenketten

◊ Kann kleinere Hindernisse (bis ca. 2cm Höhe) wie Teppichkanten, Bodenunebenheiten, auf dem Boden liegende Zeitschriften o.ä. problemlos überqueren und Rampen bis maximal etwa 30% Steigung befahren (mit der Bumper Platine ; ohne Bumper und mit maximal 2 Erweiterungsmodulen sind auch bis zu 40% Steigungen befahrbar – je nach Beschaffenheit des Untergrunds).

- **Zwei leistungsfähige MOSFET Motortreiber** (H-Brücken)

◊ Drehzahl und Drehrichtung können vom Mikrocontroller gesteuert werden

◊ Zwei **Stromsensoren** mit Messbereich bis ca. 1.8A für beide Motoren (ideal um schnell auf blockierte/stark belastete Motoren zu reagieren)

- **Zwei hochauflösende Drehgeber** zur Geschwindigkeits- und Wegmessung

◊ Auflösung von **625 CPR** ("Counts per Revolution" = Zählschritte pro Umdrehung) d.H. es werden 625 Segmente der Radencoderscheibe pro Radumdrehung von den Drehgebern gezählt! (Beim alten RP5 waren es nur etwa 4 CPR)

◊ Genaue und schnelle Geschwindigkeitsmessung und Regelung möglich!

◊ Hohe Wegstreckenauflösung von (etwa) 0.25 Millimetern pro Zählschritt!

- **Anti-Kollisions-System (Anti Collision System, ACS)** das mithilfe eines Infrarot Emfängers und zwei auf der Platine davor nach links und rechts ausgerichteten Infrarotdioden Hindernisse erkennen kann

◊ Erkennt ob sich Objekte in der Mitte, links oder rechts vor dem Roboter befinden.

◊ Die Reichweite/Sendeleistung ist in mehreren Stufen einstellbar und so können auch schlecht reflektierende Objekte erkannt werden.

- **Infrarot Kommunikationssystem (IRCOMM)**

◊ Empfängt Signale von normalen Infrarot Fernbedienungen von Fernsehern oder Videorecordern. So kann der Roboter mit einer normalen (RC5-)Fernbedienung gesteuert werden! Das Protokoll kann per Software angepasst werden, standardmäßig ist jedoch nur das gängige RC5 Protokoll implementiert.

◊ Kann zur Kommunikation von mehreren Robotern verwendet werden (Reflektion an Zimmerdecke bzw. Sichtverbindung) oder um Telemetriedaten zu übertragen.

- **Zwei Lichtsensoren** - z.B. für Helligkeitsmessungen und Lichtquellenverfolgung
- **Zwei Stoßstangensensoren (Bumper)** um Kollisionen zu erkennen
- **6 Status LEDs** - um Sensoren und Programmzustände gut darstellen zu können
  - ◊ Vier der LED Ports können auch für andere Funktionen verwendet werden!
- **Zwei freie Analog/Digital Wandler (ADC)** Kanäle für eigene Sensoren. (Diese sind auch als normale I/O Pins benutzbar).
- **Präziser 5V Spannungsregler**
  - ◊ Maximale Belastbarkeit: 1.5A
  - ◊ Große Kupferfläche zur Kühlung auf der Platine
  - ◊ Der Dauerstrom sollte ohne zusätzliche Kühlung 1A nicht überschreiten! (Es wird eine maximale Dauerlast von weniger als 800mA empfohlen)
- Leicht wechselbare **2.5A Sicherung**
- **Geringe Standby Stromaufnahme** von weniger als 5mA (und ca. 17 bis 40mA im Betrieb. Variiert je nachdem was alles angeschaltet ist (LEDs, Sensoren etc.). Diese Angabe bezieht sich natürlich nur auf die Elektronik, ohne die Motoren und ohne Erweiterungsmodule!)
- **Stromversorgung mit 6 NiMH Mignon Akkus** (nicht im Lieferumfang enthalten!)
  - ◊ z.B. Panasonic oder Sanyo (NiMH, 1.2V, 2500mAh, HR-3U , Size AA HR6) oder Energizer (NiMH, 1.2V, 2500mAh, NH15-AA)
  - ◊ Betriebszeit etwa 3 bis 6 Stunden je nach Belastung und Qualität/Kapazität der Akkus (hängt natürlich davon ab wie oft und wie lange die Motoren laufen und wie sie belastet werden! Wenn die Motoren nur selten laufen, kann der Roboter noch viel länger arbeiten. Die Angabe oben bezieht sich nur auf den Roboter ohne Erweiterungsmodule)
- **Anschluss für externe Ladegeräte** - der Hauptschalter des Roboters schaltet zwischen "Laden/Aus" und "Betrieb/An" um.
  - ◊ Dies kann über einige herausgeführte Kontakte geändert werden und so können auch externe Stromversorgungen oder zusätzliche Akkus an den Roboter angeschlossen werden.
  - ◊ Passende externe Steckerladegeräte z.B. Voltcraft 1A / 2A Delta-Peak Akkupack Schellladegerät, Ansmann ACS110, ACS410 oder AC48). Die Ladegeräte unterscheiden sich in Austattung und Ladezeit von 3 bis 14h.
- **6 kleine Erweiterungsflächen** auf dem Mainboard (und 2 sehr kleine auf der Sensorplatine) um eigene Sensorschaltungen direkt auf dem Mainboard unterzubringen. Beispielsweise könnte man rings um den Roboter noch weitere IR Sensoren anbringen um besser auf Hindernisse reagieren zu können.
  Die Erweiterungsflächen können natürlich auch für Montagezwecke verwendet werden (z.B. Drähte und Halterungen anlöten).
- **Zahlreiche Modifikationsmöglichkeiten!**

Weiterhin werden einige C Beispielprogramme sowie eine sehr umfangreiche Funktionsbibliothek mitgeliefert, welche die Programmierung stark erleichtert.

Auf der Website zum Roboter werden demnächst evtl. weitere Programme und Updates für den Roboter und seine Erweiterungsmodule zur Verfügung stehen. Sie können natürlich auch gerne Ihre eigenen Programme über das Internet mit anderen RP6 Anwendern austauschen! Die RP6Library und die Beispielprogramme stehen unter der Open Source Lizenz GPL!

1.4. Was kann der RP6?

Nun, direkt aus der Verpackung genommen noch nicht viel!

Erst durch seine Programmierung kann der RP6 etwas tun - was das genau ist, liegt komplett bei Ihnen und Ihrer Kreativität! Das macht den eigentlichen Reiz von Roboterbausätzen und ähnlichem aus: eigene Ideen umsetzen oder vorhandenes verbessern und modifizieren! Sie können aber natürlich erstmal einfach nur die vorgefertigten Programme ausprobieren und verändern um einen Eindruck von den Werksseitig gegebenen Möglichkeiten zu bekommen.

Hier werden also nur einige wenige Beispiele genannt - es liegt ganz bei Ihnen was Sie aus dem RP6 machen - es gibt noch viel mehr Möglichkeiten (s. nächste Seite)!

Der RP6 kann von Haus aus z.B. ...:

- ... autonom (d.H. selbstständig *ohne* Fernsteuerung o.ä.) umherfahren
- ... Hindernissen ausweichen
- ... Lichtquellen suchen/verfolgen
- ... Kollisionen, blockierte Motoren und niedrigen Batterie-Ladezustand erkennen und darauf reagieren
- ... die Fahrgeschwindigkeit messen und automatisch einregeln – nahezu unabhängig vom Ladezustand der Batterien, dem Gewicht des Roboters etc. (das ist die Hauptanwendung der hochauflösenden Encoder)
- ... bestimmte Distanzen fahren, sich um bestimmte Winkel drehen und die zurückgelegte Wegstrecke bestimmen (allerdings mit Abweichungen, s. Kapitel 2)
- ... bestimmte Muster und Figuren abfahren wie Kreise, bel. Vielecke, o.ä.
- ... mit anderen Robotern oder Geräten über das Infrarot-Kommunikationssystem Daten austauschen oder darüber gesteuert werden. Das funktioniert mit gängigen TV/Video/HiFi Fernbedienungen! Damit kann man den Roboter z.B. ähnlich wie ein ferngesteuertes Auto bedienen.
- ... Sensorwerte und andere Daten über das USB Interface an den PC übertragen
- ... sehr gut und einfach über das Bussystem erweitert werden! Er lässt sich einfach mit vielen weiteren Fähigkeiten ausstatten!
- ... modifiziert und den eigenen Vorstellungen angepasst werden. Einfach mal in die Schaltpläne auf der CD schauen und die Platine etwas genauer betrachten! Aber ändern Sie bitte nur etwas, wenn Sie wissen was Sie da eigentlich tun! Sie sollten lieber zunächst mit einem der Erweiterungsboards anfangen – vor allem wenn Sie noch nie eine Schaltung zusammengelötet haben...

1.5. Anwendungsvorschläge und Ideen

Der RP6 ist auf Erweiterung seiner Fähigkeiten ausgelegt - mit Erweiterungsmodulen und zusätzlichen Sensoren könnte man dem RP6 z.B. folgende Dinge „beibringen" (einige der hier genannten Aufgaben sind schon recht kompliziert und nicht ganz so einfach umzusetzen, die Themen sind *grob* aufsteigend nach Schwierigkeit geordnet):

- Den Roboter mit weiteren Controllern und damit mit zusätzlicher Rechenleistung, Speicher, I/O Ports und A/D Wandlern etc. ausrüsten. Oder wie in den Beispielprogrammen kurz angesprochen mit einfachen I²C Porterweiterungen und A/D Wandlern erweitern.
- Sensordaten und Texte auf einem LC-Display auf dem Roboter ausgeben
- Auf Geräusche reagieren und Töne erzeugen
- Mit zusätzlichen Objektsensoren, Ultraschallsensoren, Infrarotsensoren o.ä. die Distanz zu Hindernissen bestimmen und diesen somit besser ausweichen
- Schwarze Linien auf dem Boden verfolgen
- Andere Roboter oder Gegenstände verfolgen/suchen
- Den Roboter per Infrarot vom PC aus steuern (erfordert eigene Hardware – das geht leider nicht mit normalen IRDA Schnittstellen!). Oder gleich Funkmodule einsetzen.
- Den RP6 mit einem PDA oder Smartphone steuern (hier ist gemeint, dass diese Geräte auf dem Roboter montiert werden und nicht wie eine Fernbedienung o.ä. verwendet werden. Das wäre aber auch möglich!).
- Gegenstände (z.B. Teelichter oder kleine Kugeln, Metallteile ...) einsammeln
- Einen kleinen Roboterarm/Greifer anbauen um nach Gegenständen zu greifen
- Mit einem elektronischen Kompass navigieren und zusätzlich Infrarotbaken (also kleine Türme o.ä. mit vielen IR-LEDs deren Position genau bekannt ist) erkennen und so die eigene Position im Raum bestimmen und vorgegebene Zielpunkte ansteuern
- Wenn man mehrere Roboter mit einer Schussvorrichtung/Ballführung und weiteren Sensoren ausstattet, könnte man die Roboter auch Fussball spielen lassen!

- ... was immer Ihnen sonst einfällt!

Erstmal müssen Sie jedoch diese Anleitung lesen und sich grundsätzlich mit dem Roboter und der Programmierung vertraut machen. Das hier sollte nur eine kleine Motivation sein.

Und wenn es mal mit der Programmierung nicht auf Anhieb klappt, bitte nicht gleich alles aus dem Fenster werfen, aller Anfang ist schwer!

2. Der RP6 im Detail

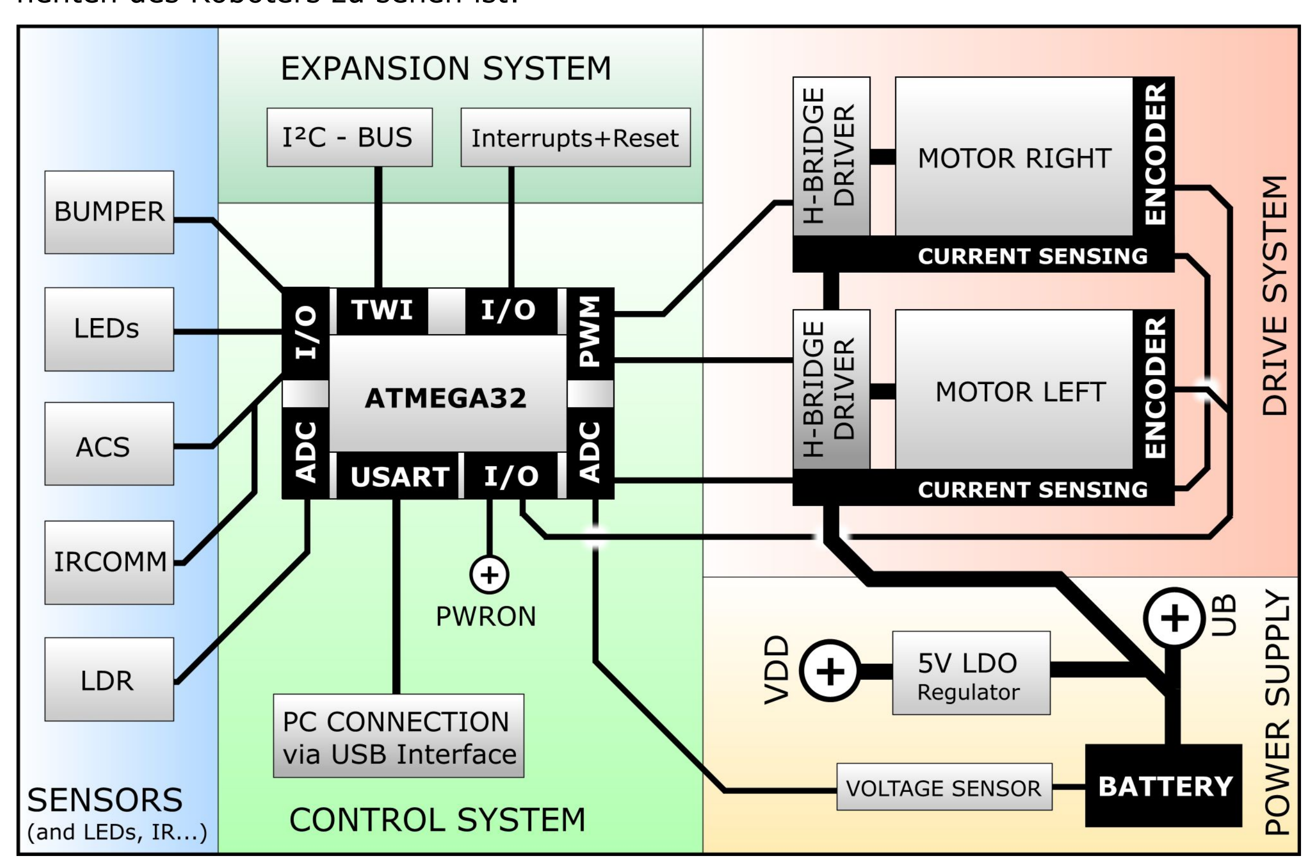
Dieses Kapitel beschäftigt sich detaillierter mit den wichtigsten Hardware Komponenten des RP6. Hier gehen wir auf die Funktionsweise der Elektronik und das Zusammenspiel mit der Software auf dem Mikrocontroller ein. Wenn Sie bereits Erfahrung mit Mikrocontrollern und Elektronik haben, brauchen Sie viele der Abschnitte in diesem Kapitel vermutlich nur kurz zu überfliegen. Allen Robotik Einsteigern empfehlen wir jedoch lieber erstmal dieses Kapitel ganz in Ruhe durchzulesen, um eine bessere Vorstellung der Funktionsweise des RP6 zu bekommen!

Wenn Sie es nicht abwarten können und lieber sofort den Roboter ausprobieren möchten, können Sie auch bei Kapitel 3 weitermachen, aber Sie sollten später noch auf dieses Kapitel zurückkommen. Es ist zum Verständnis vieler Teile der Programmierung sehr hilfreich. Sie wollen doch bestimmt gern wissen, was Sie da eigentlich mit der Software ansteuern und wie das in etwa funktioniert?

Wir gehen hier nicht allzusehr in die Tiefe, aber trotzdem könnten ein paar Dinge in diesem Kapitel auf den ersten Blick nicht ganz so leicht zu verstehen sein - der Autor hat sich aber Mühe gegeben, alles so einfach wie möglich zu erklären.

Es lohnt sich, zu verschiedenen Dingen weitere Informationen im Internet oder in Büchern zu suchen. `http://www.wikipedia.de/` ist z.B. oft ein guter Ausgangspunkt.

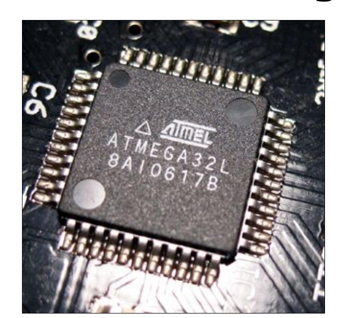
Bilder sagen bekanntlich mehr als Worte, also fangen wir mit einem Blockdiagramm des RP6 an, auf dem eine stark vereinfachte Darstellung der elektronischen Komponenten des Roboters zu sehen ist:



Man kann den RP6 grob in 5 Funktionsgruppen unterteilen:

- Steuerungssystem (Control System)
- Stromversorgung (Power Supply)
- Sensorik, IR Kommunikation und Anzeigen (Sensors) – alles was mit der Aussenwelt kommunizieren kann bzw. bestimmte Werte misst.
- Antriebssystem (Drive System)
- Erweiterungssystem (Expansion System)

2.1. Steuerungssystem



Wie man auf dem Blockdiagramm leicht erkennen kann, ist das zentrale Element des Roboters der ATMEGA32 8-Bit Mikrocontroller von ATMEL (s. Abb.).

Ein Mikrocontroller ist ein vollständiger kleiner Computer, der auf einem einzigen Chip integriert ist. Der Unterschied zu dem großen Computer vor dem Sie vermutlich gerade sitzen ist, dass er von allem sehr viel weniger hat bzw. einige Dinge gleich weggelassen wurden. Er verfügt natürlich über keine riesige Festplatte, oder mehrere Gigabyte RAM! Ein Mikrocontroller kommt auch mit weit weit weniger aus. Der MEGA32 hat z.B. "nur" 32KB (32768 Bytes) Flash ROM - das ist quasi seine "Festplatte". Darin werden alle Programmdaten gespeichert. Sein Arbeitsspeicher ist 2KB (2048 Bytes) groß und damit für unsere Zwecke mehr als ausreichend. (Zum Vergleich: Der Controller auf dem alten RP5 hatte gerade mal 240 Bytes RAM wovon fast alles vom Basic Interpreter verwendet wurde)

Und wie kann der Mikrocontroller mit so wenig Speicher auskommen? Ganz einfach: Er verarbeitet keine großen Datenmengen und braucht weder ein Betriebssystem wie Linux oder gar Windows, noch eine graphische Oberfläche oder sonstiges in der Art. Es läuft nur ein einziges Programm auf dem Controller, nämlich unser eigenes!

Das ist keineswegs ein Nachteil, sondern einer der größten Vorteile eines Mikrocontrollers gegenüber einem großen Computer (neben Energieverbrauch, Platzbedarf und Kosten)! Er kann sehr zeitkritische Dinge erledigen, die auf die Mikrosekunde genau ausgeführt werden müssen. Man kann meist genau ermitteln, wie lange er zur Ausführung eines bestimmten Programmteils braucht, denn man teilt sich die verfügbare Rechenleistung nicht mit vielen anderen Programmen wie auf einem normalen PC.

Der Controller auf dem RP6 wird mit 8MHz getaktet und führt so 8 Millionen Instruktionen pro Sekunde aus. Es wären zwar bis zu 16MHz möglich, darauf wird jedoch aus Stromspargründen verzichtet - er ist auch so schnell genug für die Aufgaben die er für gewöhnlich übernimmt! (Wieder der Vergleich mit dem alten RP5: Dieser hat bei 4MHz Takt nur ca. 1000 Basic Befehle pro Sekunde ausgeführt. Daher musste u.a. die ACS Steuerung von einem weiteren kleinen Controller übernommen werden – was nun nicht mehr notwendig ist!) Wer noch mehr Rechenleistung benötigt, kann einen oder mehrere zusätzliche Controller auf Erweiterungsmodulen anschließen. Das separat erhältliche RP6Control bietet u.a. einen zweiten MEGA32 der mit den maximal möglichen 16MHz getaktet wird.

Mit der Aussenwelt kommuniziert der Controller über seine 32 I/O Pins ("Input/Output Pins" also Ein- und Ausgangs Pins). Die I/O Pins sind in sog. "Ports" zu jeweils 8 I/O Pins organisiert. Davon hat der MEGA32 also 4 Stück: PORTA bis PORTD.

Der Controller kann den logischen Zustand der Ports einlesen und in der Software weiterverarbeiten. Genauso kann er auch logische Zustände über die Ports ausgeben und kleinere Lasten wie z.B. LEDs damit schalten (bis ca. 20mA).

Zusätzlich verfügt der Controller über viele integrierte Hardware Module, die spezielle Funktionen übernehmen können. Diese wären meist gar nicht oder nur mit viel Aufwand in Software realisierbar und würden viel zuviel wertvolle Rechenzeit in Anspruch nehmen. So hat er z.B. drei verschiedene "Timer", die u.a. Taktzyklen zählen können und deshalb häufig für Zeitmessungen eingesetzt werden. Die Timer laufen unabhängig von der Programmausführung des Mikrocontrollers und dieser kann während er z.B. auf einen bestimmten Zählerstand wartet, andere Dinge tun. Einer der Timer wird beim RP6 dazu verwendet, zwei Pulsbreitenmodulierte Signale (PWM="Pulse Width Modulation") zu erzeugen, mit denen die Spannung der Motoren und damit deren Drehzahl eingestellt werden kann. Nachdem man einen der Timer im Programm einmal wie gewünscht konfiguriert hat, arbeitet er anschließend ohne weiteres zutun fortlaufend im Hintergrund. Mehr zum Thema PWM folgt im Abschnitt "Antriebssystem".

Einige andere Module des MEGA32 sind z.B.:

- Eine serielle Schnittstelle (UART) die zur Kommunikation mit dem PC über das USB Interface verwendet wird. Hier könnte man auch andere Mikrocontroller mit einem UART anschließen - solange das USB Interface nicht angeschlossen ist!
- Das "TWI" (="Two Wire Interface", also Zweidraht Schnittstelle) Modul für den I²C Bus des Erweiterungssystems.
- Ein Analog/Digital Wandler ("Analog to Digital Converter", ADC) mit 8 Eingangskanälen, der Spannungen mit 10bit Auflösung messen kann. Damit werden auf dem RP6 die Batteriespannung überwacht, die Motorstromsensoren ausgewertet und die Lichtstärke über zwei lichtabhängige Widerstände gemessen.
- Drei externe Interrupt Eingänge. Diese erzeugen ein sog. Interrupt Ereignis, das den Programmablauf im Controller unterbricht und ihn zu einer speziellen "Interrupt Service Routine" springen lässt. Er arbeitet diese dann zunächst ab und kehrt danach wieder an die Stelle im Programm zurück, an der er sich vor dem Interrupt Ereignis befand. Das wird z.B. für die Drehgeber verwendet. Aber dazu später mehr...

Diese Spezialfunktionen können alternativ zu den normalen I/O Pins geschaltet werden, sie haben also keine eigenen Pins am Mikrocontroller. Welche der Funktionen aktiv sind, kann man einstellen. Da auf dem RP6 aber fast alles fest verdrahtet ist, macht es natürlich kaum einen Sinn, die Standardbelegung zu ändern.

Der MEGA32 hat noch viele andere Dinge integriert, aber die können wir hier nicht alle explizit beschreiben. Wer mehr wissen will, kann sich im Datenblatt des Herstellers schlau machen (auf der CD).

Man muss dazu natürlich die englische Sprache beherrschen! Fast jede Dokumentation von Drittherstellern ist nur in Englisch verfügbar. Das ist normal, denn die englische Sprache ist Standard in der Elektronik und Computer Branche.

Tipp: Unter `http://dict.leo.org/` *finden Sie ein sehr gutes, kostenlos verwendbares online Deutsch/Englisch Wörterbuch, das auch viele technische Begriffe kennt.*

2.1.1. Bootloader

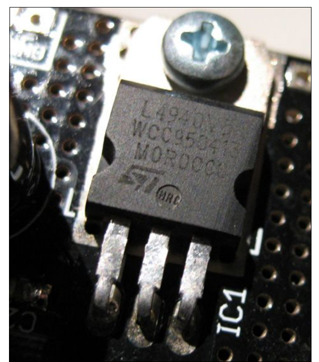
Im Mikrocontroller befindet sich in einem speziellen Speicherbereich der sog. Bootloader. Das ist ein kleines Programm, das über die serielle Schnittstelle des Mikrocontrollers neue Programme in den Programmspeicher laden kann. Der Bootloader kommuniziert mit einem Host PC über die mitgelieferte RP6Loader Software. Durch den speziell für den RP6 entwickelten Bootloader entfällt ein normalerweise benötigtes Programmiergerät. Einziger kleiner Nachteil: Es sind von den 32KB Programmspeicher des MEGA32 nur 30KB für Sie nutzbar! Das macht aber nichts – auch das reicht schon für sehr komplexe Programme (zum Vergleich: der kleinere Roboter ASURO von AREXX hat nur 7KB freien Speicher und trotzdem kommt man gut damit aus)!

2.2. Stromversorgung

Natürlich benötigt ein Roboter Energie. Diese trägt der RP6 gespeichert in 6 Akkus mit sich herum. Die Laufzeit ist durch die Kapazität der Akkus eingeschränkt, denn auch wenn die Elektronik verhältnismäßig wenig Energie benötigt, schlucken die Motoren je nach Belastung doch ziemlich viel.

Damit die Akkus möglichst lange halten und der Roboter nicht ständig Pause machen muss, sollte man ihm daher etwas größere Energiespeicher mit etwa 2500mAh gönnen. Kleinere mit 2000mAh oder mehr funktionieren aber auch. Mit guten Akkus kann man 3 bis 6 Stunden oder mehr Laufzeit erreichen, abhängig von der Betriebszeit der Motoren, deren Belastung und Qualität der Akkus. Die 6 Akkus die notwendig sind, haben insgesamt eine Nennspannung von 6x 1.2V = 7.2V. Im Blockdiagramm ist dies mit "UB" (= "U-Battery", U ist der Formelbuchstabe für Spannung) bezeichnet. „Nennspannung", weil sich die Spannung mit der Zeit stark verändert. Voll aufgeladen können die Akkus im Leerlauf insgesamt bis zu 8.5V liefern! Diese Spannung sinkt während der Entladung ab und schwankt auch je nach Belastung (Motoren an oder aus, schnell langsam etc. - wie stark die Spannung schwankt, hängt auch von der Qualität der verwendeten Akkus ab. Der Innenwiderstand ist hier die kritische Größe).

Das ist für Messungen von Sensorwerten und ähnlichem natürlich nicht ohne weiteres brauchbar. Noch wichtiger ist jedoch, dass viele der verwendeten Komponenten wie z.B. der Mikrocontroller nur auf 5V oder weniger Betriebsspannung ausgelegt sind und bei so hohen Spannungen zerstört würden. Die Akku Spannung muss also auf einen definierten Wert heruntergeregelt und stabilisiert werden!



Das übernimmt ein 5V Spannungsregler, der einen Strom von maximal 1.5A liefern kann (s. Abb.). Bei 1.5A würde er jedoch ziemlich viel Wärme abgeben. Es gibt daher eine große Kühlfläche auf der Platine an die der Regler festgeschraubt ist. Über 1A sollte der Regler trotz Kühlung besser nur kurzzeitig (also wenige Sekunden) belastet werden, wenn man nicht noch einen zusätzlichen großen Kühlköper draufschraubt.

Es wird empfohlen, 800mA Dauerlast nicht zu überschreiten! Bei so einer Belastung und zusammen mit den Motoren wären die Akkus auch recht schnell leer. Im normalen Betrieb ohne ein Erweiterungsboard nimmt der Roboter übrigens nicht mehr als 40mA auf (ausser wenn das IRCOMM sendet), also überhaupt kein Problem für den Regler und man hat noch genug Reserven für die Erweiterungsmodule, die meistens auch nicht mehr als 50mA aufnehmen sofern keine Motoren, große LEDs o.ä. daran angeschlossen sind.

2.3. Sensorik

Die meisten Sensoren über die der Roboter verfügt, haben wir ja schon in anderen Abschnitten kurz genannt, wollen diese nun aber etwas detaillierter betrachten.

In dem Blockdiagramm sind einige der Sensoren nicht in dem blauen Bereich "Sensors" zu sehen, weil sie besser in andere Bereiche passen. Trotzdem zählen natürlich auch die Drehgeber (="Encoder"), Motorstromsensoren und der Batteriespannungssensor zu den Sensoren und werden darum in diesem Abschnitt beschrieben!

2.3.1. Batteriespannungs-Sensor (Voltage Sensor)

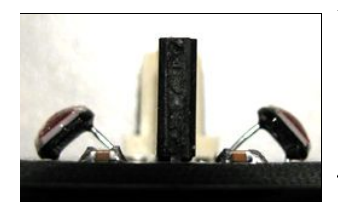
Dieser "Sensor" ist eigentlich nur ein einfacher Spannungsteiler aus zwei Widerständen. Wir gehen davon aus, dass die Batterien insgesamt maximal 10V liefern können. 6 NiMH Akkus werden immer unterhalb davon bleiben. Die Referenzspannung des ADC, also die Spannung mit der er die gemessene Spannung vergleicht, beträgt 5V. Da auch die Betriebsspannung 5V beträgt, darf diese nicht überschritten werden. Also müssen wir die zu messende Spannung um die Hälfte herabsetzen! Dies geschieht über einen Spannungsteiler aus zwei Widerständen, der die Spannung an den Messbereich des ADCs anpasst.

Der ADC löst mit 10 Bit auf (Wertebereich 0 bis 1023), was eine Auflösung von $\frac{10V}{1024} = 9.765625mV$ ergibt. Ein Messwert von 512 entspricht hier also 5V und 1023 wären ungefähr 10V! Diese Grenzwerte sollten aber normalerweise nie erreicht werden!

Das ist nicht besonders genau, da die Widerstände keineswegs Präzisionswiderstände sind. Abweichungen von einigen Prozent nach oben und unten sind möglich. Auch die Referenzspannung von 5V ist nicht ganz genau und kann bei normaler Belastung etwas variieren. Das stört hier nicht, denn wir brauchen ohnehin nur einen ungefähren Wert um festzustellen ob die Batterien sich langsam dem Ende nähern. Wer die Spannung genau messen will, braucht ein Multimeter um den Messfehler zu ermitteln und diesen dann in Software auszugleichen.

Wenn man mit Fehlern leben kann, kann man die Spannung dank des günstigen Umrechnungsverhältnisses sogar direkt aus den ADC Werten ungefähr abschätzen: 720 entsprechen dann grob 7.2V, 700 etwa 7V und 650 wären etwa 6.5V. Bei einem konstanten Wert unter 560 kann man davon ausgehen, dass die Akkus fast leer sind.

2.3.2. Lichtsensoren (LDRs)



Vorne auf der kleinen Sensorplatine des Roboters sind zwei sog. LDRs (="Light Dependant Resistors" also lichtabhängige Widerstände) platziert und nach links bzw. rechts ausgerichtet. Zwischen den beiden Sensoren ist noch eine kleine schwarze „Trennwand" damit das Licht aus einer Richtung möglichst nur einen der Sensoren erreicht. Sie bilden zusammen mit je einem normalen Widerstand wie beim Batteriesensor einen Spannungsteiler – hier allerdings um das Umgebungslicht zu messen. Die 5V Betriebsspannung wird auch geteilt, aber hier ist einer der Widerstände variabel! Es wird also das Teilungsverhältnis je nach Intensität des Lichteinfalls verändert und somit eine vom Lichteinfall abhängige Spannung an einen der A/D Wandler Kanäle geleitet!

Über den Spannungsunterschied zwischen den beiden Sensoren kann man ermitteln in welcher Richtung sich eine bzw. die hellste Lichtquelle vor dem Roboter befindet: Links, Rechts oder in der Mitte.

Mit einem entsprechenden Programm, kann man so z.B. eine starke Taschenlampe in einem *abgedunkelten* Zimmer verfolgen, oder den Roboter die hellste Stelle in einem Raum suchen lassen! Klappt z.B. sehr gut mit einem stärkeren Hand-Halogenscheinwerfer: Wenn man damit auf den Boden leuchtet, kann der Roboter dem Lichtfleck auf dem Boden folgen.

Das geht natürlich auch umgekehrt: Der Roboter könnte z.B. auch versuchen sich vor hellem Licht zu verstecken...

Wenn man noch ein oder zwei LDRs hinten am Roboter anbringen würde, könnte man das noch verfeinern und die Richtung in der sich Lichtquellen befinden besser bestimmen. Der Roboter kann sonst nämlich oft nur schwer unterscheiden ob die Lichtquelle vor oder hinter ihm liegt. Zwei der A/D Wandler Kanäle sind noch frei...

2.3.3. Anti Collision System (ACS)

Der aus Softwaresicht komplexeste Sensor des RP6 ist das ACS - das „Anti Collision System" (engl. für Anti Kollisions System)! Es besteht aus einem IR Empfänger (s. Abb.) und zwei vorne auf der Sensorplatine links und rechts angebrachten IR LEDs. Die IR LEDs werden direkt vom Mikrocontroller angesteuert. Die Ansteuerungsroutinen können beliebig verändert und verbessert werden! Beim Vorgängermodell war dafür noch ein eigener Controller nötig, dessen Programm aber nicht vom Anwender geändert werden konnte...

Mit den IR LEDs werden kurze, mit 36kHz modulierte Infrarot Impulse ausgesandt, auf die der darauf ausgelegte IR Empfänger reagiert. Werden die IR Impule an einem Gegenstand vor dem Roboter reflektiert und vom IR Empfänger detektiert, kann der Mikrocontroller darauf reagieren und z.B. ein Ausweichmanöver einleiten. Damit das ACS nicht zu empfindlich bzw. auf eventuelle Störungen reagiert, wartet die Software bis eine bestimmte Anzahl von Impulsen in einer bestimmten Zeit empfangen worden ist. Es wird auch eine Synchronisation mit dem RC5 Empfang durchgeführt und auf die RC5 Signale von TV/Hifi Fernbedienungen wird so nicht reagiert. Bei anderen Codes kann das aber nicht garantiert werden und das ACS könnte dann Hindernisse erkennen, wo gar keine sind!

Da es je eine IR LED links und rechts gibt, kann das ACS grob unterscheiden ob sich das Objekt links, rechts oder mittig vor dem Roboter befindet.

Man kann zusätzlich noch die Stromstärke mit der die beiden IR LEDs gepulst werden in drei Stufen einstellen. Das ACS funktioniert aber auch in der höchsten Stufe nicht mit allen Objekten immer zuverlässig, denn es kommt auf die Beschaffenheit der Oberfläche des jeweiligen Objekts an! Ein schwarzes Objekt reflektiert das IR Licht natürlich viel schlechter als ein weisses Objekt und ein kantiges und spiegelndes Objekt könnte das IR Licht hauptsächlich in eine bestimme Richtung reflektieren. Die Reichweite ist also immer vom jeweiligen Objekt abhängig! Das ist eine prinzipielle Schwäche von so gut wie allen IR Sensoren (jedenfalls in dieser Preisklasse).

Trotzdem werden die meisten Hindernisse zuverlässig erkannt und können umfahren werden. Falls das mal nicht klappt, gibt es noch die Stoßstange mit den Tastsensoren und falls auch die nicht richtig getroffen werden, kann der Roboter noch mit den Motorstromsensoren oder den Encodern erkennen, ob die Motoren blockieren (s.u.)!

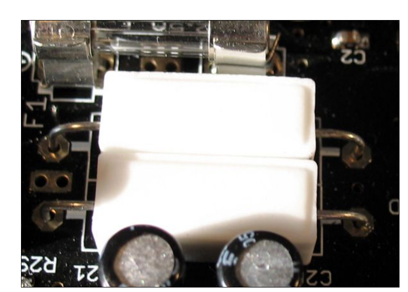
Wem das nicht reicht, der könnte zusätzlich z.B. noch Ultraschallsensoren anbringen...

2.3.4. Stoßstangensensoren (Bumper)

Vorn am Roboter sind zwei Mikroschalter mit langem Schalthebel auf einer separaten Platine untergebracht, die etwas vor der anderen Sensorplatine liegt. Dadurch werden die IR LEDs auf der Sensorplatine geschützt und können nicht so leicht verbiegen wenn der Roboter mal gegen ein Hindernis fährt. Mit den zwei Schaltern kann der Mikrocontroller einen solchen Aufprall registrieren und dann beispielsweise zurücksetzen, sich etwas drehen und weiterfahren. Die Schalter sind an zwei der Ports die schon mit den LEDs verbunden sind angeschlossen und blockieren so keine anderen Ports des Mikrocontrollers. Daher leuchten die LEDs auch immer wenn man einen der Schalter drückt! Da dies normalerweise relativ selten passiert, stört das aber nicht weiter.

Die Stoßstange kann man auch abmontieren und z.B. gegen eine Schuss- oder Sammelvorrichtung für Bälle o.ä. ersetzen wenn man möchte.

2.3.5. Motorstromsensoren (Current sensing)



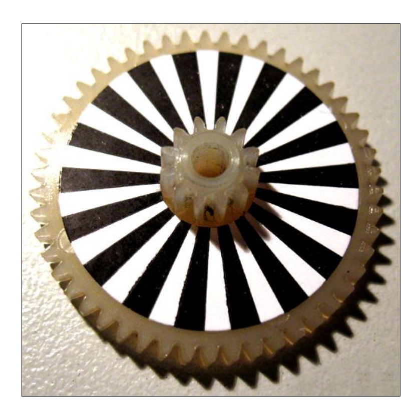
Es befinden sich zwei Leistungswiderstände in den beiden Motorstromkreisen. Aus dem Ohm'schen Gesetz $U = R \cdot I$ folgt, dass sich die Spannung die an einem bestimmten Widerstand abfällt, proportional zum Strom verhält, der diesen durchfließt!

Damit die Spannungsabfälle an den Widerständen nicht zu groß werden, müssen die Widerstände klein gewählt werden. In unserem Fall haben Sie einen Wert von 0.1 Ohm

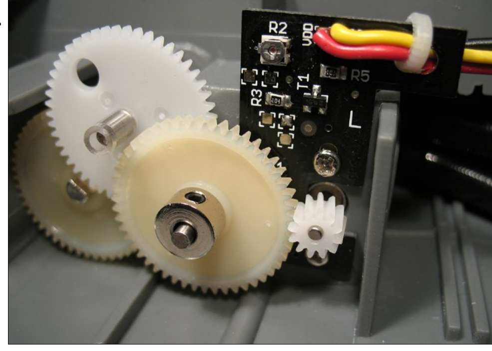
Die abfallende Spannung ist also nur sehr klein (0.1V bei einem Strom von 1A) und muss verstärkt werden bevor sie mit dem ADC gemessen werden kann. Das erledigt jeweils ein sog. Operationsverstärker (OPV). In der Schaltung des RP6 wird je Motorkanal ein OPV verwendet. Der Messbereich geht etwa bis 1.8A. Bei 1.8A fallen etwa 0.18V am Widerstand ab und es ergibt sich am Ausgang des OPV eine Spannung von etwa 4V. Mehr kann der verwendete OPV bei 5V Betriebsspannung nicht ausgeben.

Die Leistungswiderstände haben eine Toleranz von 10%, die Widerstände am OPV 5%, also ist das alles nur sehr ungenau (Ungenauigkeiten im Bereich von etwa 270mA sind möglich wenn man die Sensoren nicht kalibriert!). Wir brauchen allerdings auch nur den ungefähren Wert um festzustellen ob die Motoren stark oder wenig belastet werden. So kann der Roboter gut blockierte oder gar defekte Motoren bzw. Drehgeber erkennen! Die DC-Motoren benötigen mehr Strom je stärker sie belastet werden (Drehmoment) und somit steigt der Strom sehr stark an wenn die Motoren blockiert sind. Das wird von der Robotersoftware zur Notabschaltung verwendet: wenn die Motoren dauerhaft mit hohem Strom betrieben würden, könnten diese sehr heiß werden und dadurch Schaden nehmen! Und wenn die Encoder mal ausfallen sollten – aus welchem Grund auch immer – kann auch das damit erkannt werden. Man würde dann eine Drehzahl von 0 messen. Lässt man die Motoren aber auf voller Kraft laufen und der Strom bleibt trotzdem klein (also sind die Ketten nicht blockiert!), kann man genau daraus schließen, dass entweder die Encoder, oder die Motoren ausgefallen sind (oder Encoder und Motorstromsensoren nicht funktionieren... das passiert z.B. wenn man vergessen hat diese vorher per Software einzuschalten).

2.3.6. Drehgeber (Encoder)



Ganz anders als die letztgenannten Sensoren funktionieren die Drehgeber, die an den Getrieben der Motoren zur Drehzahlmessung angebracht sind. Es handelt sich dabei um Reflexlichtschranken, die auf Codierscheiben mit je 18 weissen und 18 schwarzen Segmenten ausgerichtet sind, also insgesamt 36 Segmente (s. Abb). Diese Codierscheiben sind wiederum an je eines der Zahnräder der beiden Getriebe geklebt worden. Wenn es sich dreht, wandern die einzelnen Segmente an der Reflexlichtschranke vorbei. Die weissen Segmente reflektieren das Infrarotlicht, die schwarzen Segmente nur wenig. Die Drehgeber erzeugen so zwar auch wie die anderen Sensoren ein analoges Signal, aber es wird digital interpretiert. Zunächst wird das Signal verstärkt und anschließend über einen sog. Schmitt Trigger in ein Rechtecksignal umgewandelt. Die Flanken dieses Signals, also die Wechsel von 5 auf 0V und umgekehrt, lösen jeweils ein Interrupt Ereignis aus und diese werden dann von der Software gezählt. So kann die zurückgelegte Wegstrecke gemessen und zusammen mit einem Timer zur Zeitmessung die Drehzahl und damit auch die Geschwindigkeit ermittelt werden. Die Ermittlung der Drehzahl ist auch Hauptanwendung der Encoder – nur mit den Encodern kann man die Drehzahl auf den gewünschten Sollwert einregeln. Ohne Regelung wäre die Drehzahl nämlich von der Akkuspannung, Belastung der Motoren usw. abhängig. Die hohe Auflösung ermöglicht es dabei, auch niedrige Geschwindigkeiten noch relativ genau einzuregeln.



Jedes der zwei mittleren Stufenzahnräder des Getriebes hat 50 Zähne auf dem großen, und 12 auf dem kleineren Zahnrad (s. Abb). Die Codierscheiben befinden sich auf dem Zahnrad neben dem Motor, also rechnet man:

$$\frac{50}{12}\cdot\frac{50}{12}=17\frac{13}{36}\;\;;\;\;17\frac{13}{36}\cdot\mathbf{36}=625$$

Daher haben die Encoderscheiben auch Ihre 36 Segmente, denn das gibt eine schöne runde Zahl ohne gebrochenen Anteil. Die Drehgeber erzeugen also 625 Flanken pro Radumdrehung wobei jede Flanke einem Segment entspricht.

Bei einem Raddurchmesser von ca. 50mm inkl. Raupenkette, ergibt sich rein rechnerisch ein Radumfang von ca. 157mm was 0.2512mm pro Zählschritt der Drehgeber entspricht. Da sich die Raupenketten aber fast immer etwas in den Untergrund eindrücken (bzw. auch selbst eingedrückt werden) kann man aber von 0.25mm pro Zählschritt ausgehen – meist ist es sogar etwas weniger, z.B. nur 0.24 oder 0.23mm. Das muss man durch abfahren von Teststrecken ermitteln, wie es im Anhang grob beschrieben ist. Sehr genau ist das allerdings durch Radschlupf (bzw. hier müssten wir eigentlich von „Kettenschlupf" sprechen) und ähnlichen Dingen nicht - vor allem beim Rotieren auf der Stelle. Beim normalen Geradeausfahren ist dieser Fehler klein, aber beim Rotieren kann er schnell sehr große Werte annehmen! Abweichungen muss man evtl. durch weitere Tests ermitteln und mit einkalkulieren. Das ist bei Raupenantrieben leider so – auch bei viel teureren und hochwertigeren Robotern. Dafür hat man aber den Vorteil, dass der Roboter mit Raupenantrieb recht geländegängig ist im Vergleich zu Robotern mit normalem Differentialantrieb mit zwei Antriebsrädern und Stützrad. Kleinere Hindernisse und Rampen kann er meist problemlos überwinden.

Dabei sind die Encoder sehr nützlich, denn man kann die Geschwindigkeit gut einregeln, egal wie der Untergrund und die Motorbelastung gerade aussieht.

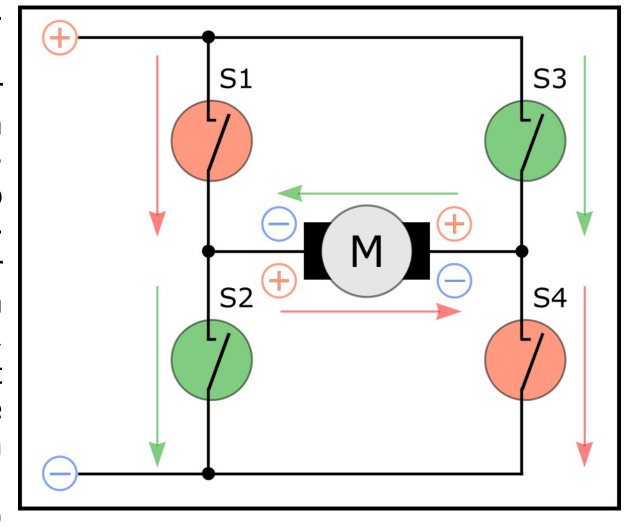
Bei gemessenen 50 Segmenten pro Sekunde liegt die Geschwindigkeit bei 1.25 cm/s, sofern wir von 0.25mm pro Zählschritt ausgehen. Etwa 50 Segmente pro Sekunde ist auch die geringste gerade noch regelbare Geschwindigkeit (das variiert aber etwas von Roboter zu Roboter). Bei 1200 Segmenten pro Sekunde wären es die maximal möglichen 30 cm/s (bei 0.25mm Auflösung, bei 0.23 sind es 27.6 cm/s). Standardmäßig begrenzt die Funktionsbibliothek das aber auf 1000 Flanken pro Sekunde. Die maximale Geschwindigkeit ist vom Ladezustand der Akkus abhängig – daher wären 30cm/s nicht besonders lange haltbar. Außerdem erhöht sich die Lebensdauer der Getriebe und Motoren je langsamer man fährt!

Wenn der Roboter 4000 Segmente gezählt hat, ist er übrigens etwa einen Meter weit gefahren. Aber wie schon gesagt, gilt das natürlich nur für genau 0.25mm Auflösung. Ohne Kalibrierung hat man hier immer mehr oder weniger starke Abweichungen. Wem es nicht auf jeden mm ankommt, braucht nichts zu kalibrieren und kann einfach von 0.25mm oder besser 0.24mm ausgehen!

Optimal ist es, wenn man sich für Weg- und Winkelmessungen nicht auf die Encoder Daten stützen muss, sondern externe Systeme wie Infrarotbaken oder einen genauen elektronischen Kompass dazu zur Verfügung hat.

2.4. Antriebssystem

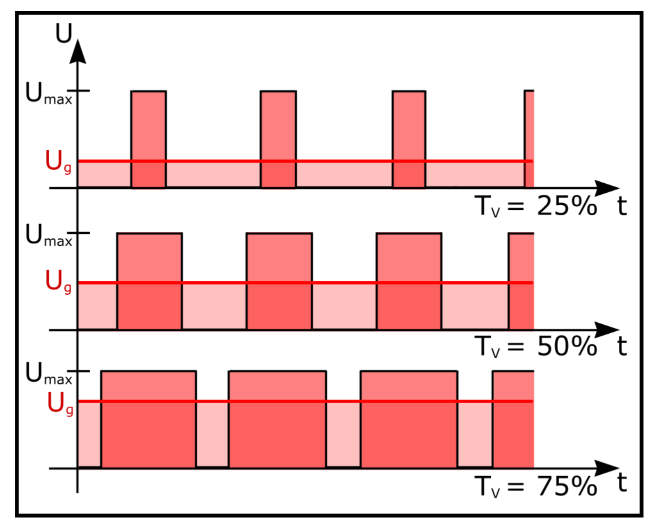
Der Antrieb des RP6 besteht aus zwei Gleichstrom Motoren mit nachgeschaltetem Getriebe, über das die beiden Raupenketten angetrieben werden (s. Abb. weiter oben).



Die Motoren genehmigen sich je nach Belastung einen recht hohen Strom und können natürlich nicht direkt vom Mikrocontroller angesteuert werden. Dazu braucht man Leistungstreiber in Form von je einer H-Brücke pro Motor. Das grundlegende Prinzip ist in der nebenstehenden Abbildung dargestellt. Eine H-Brücke besteht aus vier "Schaltern", die in Form eines H's um einen Motor angeordnet sind. Nehmen wir mal an, zunächst seien alle Schalter aus. Schaltet man dann S1 und S4 (Rot) an, liegt eine Spannung am Motor an und er dreht sich z.B. nach rechts. Schalten wir nun S1 und S4 wieder aus und danach S2 und S3 (Grün) an, wird die am Motor anliegende Spannung umgepolt und er dreht sich in die entgegengesetzte Richtung, also nach links! Man muss darauf achten, nicht gleichzeitig S1 und S2, oder S3 und S4 einzuschalten, sonst entstünde ein Kurzschluss und dadurch könnten die "Schalter" zerstört werden!

Natürlich verwenden wir auf dem RP6 keine mechanischen Schalter, sondern sog. MOSFETs. Diese schalten beim Anlegen einer genügend hohen Spannung am Eingang durch. Die Schaltvorgänge können sehr schnell erfolgen, mehrere Kilohertz sind bei unserer Anwendung möglich.

So kann man also schonmal die Drehrichtung einstellen. Und wie bekommt man den Motor nun schneller bzw. langsamer? Ein Elektromotor dreht umso schneller, je höher die angelegte Spannung ist. Die Drehzahl kann also über die Spannung eingestellt werden - und genau dazu können wir die H-Brücke auch verwenden!

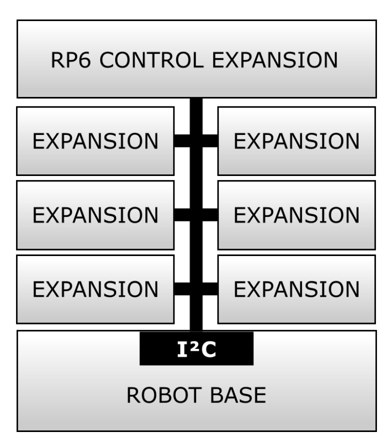


Die Abbildung verdeutlicht das Prinzip nach dem wir vorgehen können. Wir erzeugen ein Rechtecksignal *fester* Frequenz, bei dem wir das sog. Tastverhältnis verändern. Mit "Tastverhältnis" ist das Verhältnis von der eingeschalteten zur ausgeschalteten Zeit des Signals gemeint.

Am Motor liegt dann effektiv eine niedrigere, mittlere Gleichspannung an, die dem Tastverhältnis entspricht. In der Grafik ist dies durch eine rote Linie (U_g) und die roten Flächen verdeutlicht. Wenn z.B. eine Spannung von 7 Volt von den Akkus an den Motortreibern anliegt, und diese mit einem Tastverhältnis von 50% angesteuert werden, würde die mittlere Spannung in etwa bei der Hälfte, also 3.5V liegen! Das stimmt real nicht ganz, aber so kann man es sich schon gut und einfach vorstellen.

Das Antriebssystem ist dank der hohen Untersetzung (~ 1:72) relativ stark und so kann der RP6 viel schwerere Lasten tragen als es z.B. der kleine Roboter ASURO könnte. Allerdings steigt mit zunehmendem Gewicht natürlich auch der Energiebedarf und die Akkus werden schneller leer sein ...

Im Vergleich mit einem ferngesteuerten Rennwagen o.ä. könnte man denken der RP6 sei langsam - stimmt auch - das ist jedoch absichtlich so! Der Roboter wird von einem Mikrocontroller gesteuert und wenn der Programmierer einen Fehler bei der Programmierung gemacht hat, wäre es ungünstig mit 10km/h vor die Wand zu prallen! Bei dem eher gemächlichen Tempo des RP6 passiert aber nicht so schnell etwas und nebenbei haben die Sensoren mehr Zeit die Umgebung auf Hindernisse zu untersuchen. Und da wäre natürlich noch der Vorteil der höheren Belastbarkeit und der viel genaueren Geschwindigkeitsregelung! Der RP6 kann sehr sehr langsam bei konstanter Geschwindigkeit fahren!



2.5. Erweiterungssystem

Eines der nützlichsten Features des RP6 ist das Erweiterungssystem. Man kann den RP6 damit genau so weit ausbauen, wie man es benötigt. Aus Kostengründen bietet das Basissystem schließlich nur realtiv wenig Sensoren. Es sind zwar schon mehr als bei vielen anderen Robotern in dieser Preisklasse, aber erst mit weiteren Sensoren macht ein Roboter so richtig Spaß. Das ACS kann beispielsweise nur grob erkennen *ob* sich ein Hindernis vor dem Roboter befindet. Mit Ultraschallsensoren oder besseren zusätzlichen IR Sensoren, könnte man aber die Distanz ermitteln und so beispielsweise bessere Ausweichmanöver fahren!

Neben Sensoren macht es auch sehr viel Sinn, zusätzliche Controller einzubauen um die anfallenden Aufgaben aufteilen zu können. Beispielsweise das RP6 CONTROL M32 mit einem weiteren MEGA32 Mikrocontroller.

Das Erweiterungssystem muss natürlich in der Lage sein, viele Erweiterungsmodule miteinander zu verbinden (engl. Expansion Modules, s. Abb.), dabei mit wenigen Signalleitungen auskommen und eine ausreichend hohe Geschwindigkeit bieten.

2.5.1. Der I^2C Bus

Der I^2C Bus (sprich: „I quadrat C Bus“ --> IIC = „Inter IC Bus“), erfüllt diese Anforderungen. Er braucht nur 2 Signalleitungen, kann bis zu 127 Teilnehmer miteinander verbinden und hat eine maximale Übertragungsrate von 400kBit/s.

Der I^2C Bus wurde in den 1980er und 90er Jahren von Philips Semiconductors entwickelt und ist seitdem zu einem sehr weit verbreiteten Bussystem geworden. Der I^2C Bus findet in sehr vielen elektronischen Geräten der Unterhaltungselektronik, wie Videorecordern und Fernsehern Verwendung, aber auch in industriellen Geräten und Systemen. In den meisten moderneren PCs und Notebooks ist er in Form des SMBus zu finden und wird dort u.a. zur Kommunikation der Lüftersteuerung und Temperaturüberwachung verwendet. Auf vielen anderen Robotern wird er ebenfalls eingesetzt und so ist es nicht verwunderlich, dass es auch diverse Sensormodule wie Ultraschallsensoren, elektronische Kompassmodule, Temperatursensoren und ähnliches mit diesem Interface gibt.

Es handelt sich um einen Master-Slave orientierten Bus. Ein oder mehrere Master steuern den Datenverkehr von und zu bis zu 127 Slaves. Wir beschreiben aber nur die Verwendung des Busses mit einem Master, auch wenn der Bus Multimaster fähig wäre! Das würde die Sache nur verkomplizieren.

Die beiden Signalleitungen werden mit SDA und SCL bezeichnet. SDA steht für "Serial Data" und SCL für "Serial Clock" - also eine Daten- und eine Taktleitung. SDA ist bidirektional, was bedeutet, dass hier sowohl Master als auch Slave Daten anlegen können. Der Takt auf SCL wird immer vom Master erzeugt.

Der Bus überträgt die Datenbits stets synchron zum Taktsignal, das von einem Master vorgegeben wird. Der Pegel von SDA darf sich nur ändern während SCL low ist (ausser bei Start und Stopbedingung, s.u.). Die Übertragungsgeschwindigkeit kann auch während einer laufenden Übertragung beliebig zwischen 0 und 400kBit/s variieren.

START	ADR	W	ACK	DATA	ACK	...	DATA	ACK	STOP

START	ADR	R	ACK	DATA	ACK	...	DATA	ACK	STOP

Zwei typische Übertragungsabläufe sind in den obigen Abbildungen dargestellt. Die erste Abbildung zeigt eine Datenübertragung vom Master zu einem Slave. Die weissen Kästchen deuten auf Datenverkehr vom Master zum Slave hin, die dunklen sind die Antworten des Slaves.

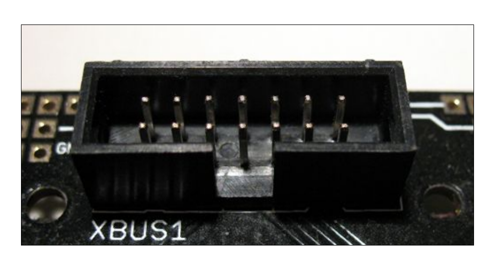
Jede Übertragung muss mit einer Startbedingung eingeleitet werden und mit einer Stopbedingung beendet werden. Die Starbedingung tritt ein, wenn bei high Pegel auf SCL die Datenleitung SDA von high auf low Pegel gezogen wird. Umgekehrt gilt dies für die Stopbedingung, also wenn während SCL high Pegel führt, SDA von low auf high gezogen wird.

Nach der Startbedingung folgt die 7Bit lange Adresse des Slaves, mit dem die Kommunikation stattfinden soll, gefolgt von einem Bit das festlegt ob Daten geschrieben oder gelesen werden. Der Slave bestätigt dies mit einem ACK ("Acknowledge" = Bestätigung). Danach folgen beliebig viele Datenbytes wobei der Empfang jedes einzelnen Bytes vom Slave mit einem ACK quittiert wird. Abgeschlossen wird die Übertragung mit der Stopbedingung.

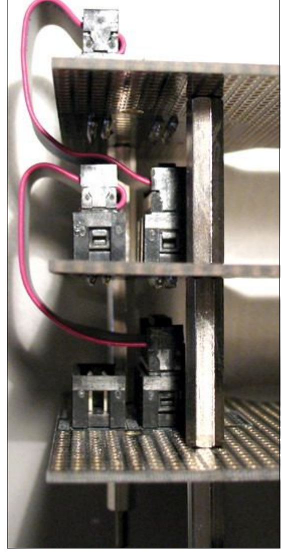
Das war hier nur eine sehr kurze Beschreibung des I²C Busses. Mehr dazu finden Sie in der I²C Bus Spezifikation von Philips. Auch im Datenblatt des MEGA32 findet sich einiges dazu!

In den Beispielprogrammen können Sie sehen, wie man das verwenden kann. In der RP6 Funktionsbibliothek sind die Funktionen zum Ansteuern des I²C Busses bereits vorhanden. Um das genaue Protokoll muss man sich also keine Gedanken mehr machen, es ist aber trotzdem nützlich, das schonmal gehört zu haben.

2.5.2. Erweiterungsanschlüsse



Auf dem Mainboard des Roboters befinden sich insgesamt vier Erweiterungsanschlüsse. Zwei davon sind mit „XBUS1" und „XBUS2" beschriftet. XBUS ist eine Abkürzung für „eXpansion BUS", also Erweiterungs-Bus. XBUS1 und XBUS2 sind vollständig miteinander verbunden und symmetrisch auf dem Mainboard angeordnet. Deshalb kann man die Erweiterungsmodule sowohl vorne, als auch hinten auf dem Roboter montieren. Auf jedem Erweiterungsmodul finden sich an einer Seite zwei XBUS Anschlüsse. Über je ein 14 poliges Flachbandkabel kann man die Erweiterungsmodule mit dem Mainboard und untereinander verbinden – dazu gibt es auch *zwei* identische und miteinander verbundene Anschlüsse auf jedem Erweiterungsmodul. Der äussere Stecker muss dabei als Verbindung nach unten genutzt werden, während der innere für die Verbindung nach oben gedacht ist. So kann man (theoretisch) beliebig viele Module übereinander stapeln (s. Abb. - hier sind drei RP6 Lochraster Erweiterungsmodule abgebildet – diese kann man für eigene Schaltungen verwenden).



Auf den XBUS Anschlüssen sind Stromversorgung, der oben beschriebene I²C-Bus, Master Reset und Interrupt Signale vorhanden.

Zur Stromversorgung stehen zwei Spannungen an den Anschlüssen bereit, natürlich die stabilisierten 5V vom Spannungsregler, aber auch die Akkuspannung. Diese verändert sich mit der Zeit und schwankt je nach Belastung – sie liegt normalerweise im Bereich von etwa 5.5 (Akkus leer) bis etwa 8.5V (frisch aufgeladene Akkus – das variiert von Hersteller zu Hersteller), kann aber auch je nach Belastung, Art und Ladezustand der Batterien zeitweilig ausserhalb dieses Bereichs liegen.

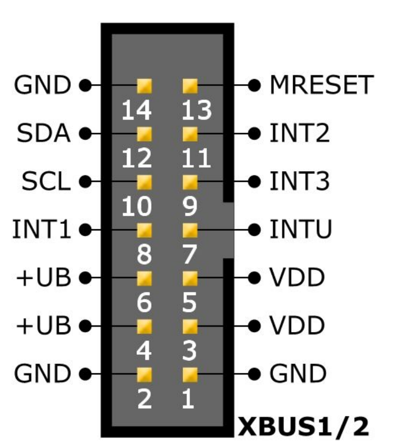
Das Master Reset Signal ist wichtig um alle Mikrocontroller beim Druck auf den Start/Stop Taster bzw. beim Programmieren zurückzusetzen. Die Bootloader in den Controllern starten ihre Programme übrigens bei einem low Puls (high-low-high) auf der SDA Leitung – so starten die Programme auf allen (AVR) Controllern gleichzeitig nachdem man den Start/Stop Taster gedrückt hat oder das Programm über die Boot-

loader Software startet... (der Bootloader generiert zum Starten allerdings nicht nur einen low Puls, sondern einen kompletten I²C General Call mit 0 als Datenbyte.)

Die Interruptleitungen werden von einigen Modulen dazu verwendet, dem Master Mikrocontroller per externem Interrupt Signal mitzuteilen, dass neue Daten vorhanden sind oder eine bestimmte Aufgabe ausgeführt worden ist und auf neue Befehle gewartet wird. Hätte man diese Leitungen nicht, müsste der Mastermikrocontroller (bei bestimmten Erweiterungsmodulen) ständig die Slaves auf neue Daten abfragen. Das ist natürlich auch möglich, aber mit zusätzlichen Interrupt Leitungen spart man sich einiges an Buslast und Rechenzeit. Da es insgesamt nur 3 Interruptleitungen und eine vom Anwender frei verwendbare Leitung gibt, müssen sich ggf. gleiche Module wie z.B. Ultraschallsensoren eine der Leitungen teilen und ggf. alle abgefragt werden.

Die beiden anderen mit „USRBUS1" und „USRBUS2" beschrifteten Erweiterungsanschlüsse auf dem Mainboard, sind NICHT miteineinander verbunden. Die einzelnen Leitungen sind auf allen Erweiterungsmodulen auf Lötpads herausgeführt, so dass man eigene Signale an diese Steckverbinder legen kann. Daher auch der Name „USRBUS" was für „User-Bus" steht, also „Anwender-Bus". Sie können diese 14 poligen Erweiterungsstecker für alles verwenden was Sie wollen – ein eigenes Bussystem, zusätzliche Versorgungsleitungen (Achtung: Nicht zu stark belasten, die Leiterbahnen sind nicht sonderlich breit) oder ähnliches. Man kann damit z.B. auch zwei der Erweiterungsmodule miteinander Verbinden, ohne diese an die anderen Erweiterungen anzuschließen. Das ist nützlich für aufwendigere Schaltungen oder Sensoren die nicht auf eines der Erweiterungsmodule passen würden... und so bleibt die Verkabelung ordentlicher.

Sie können natürlich nicht beliebig viele Erweiterungsmodule anschließen – spätestens bei 6 vorne oder hinten übereinander gestapelten Modulen dürfte der RP6 keinen Elchtest mehr bestehen. Und außerdem muss man auch auf den Energiebedarf achten! Wird dieser zu hoch, werden die Akkus sehr schnell leer sein. Pauschal kann man allerdings sagen, das insgesamt maximal bis zu 8 Erweiterungsmodule auf dem RP6 Platz finden, also jeweils 4 vorne und 4 hinten.



In der Grafik sehen Sie die Anschlussbelegung der beiden Erweiterungsstecker. Pin 1 liegt auf dem Mainboard immer auf der Seite auf der sich die weisse Beschriftung XBUS1 bzw. XBUS2 befindet bzw. ist mit „1" neben dem Stecker beschriftet .

+UB ist die Batteriespannung, VDD die +5V Betriebsspannung, GND bezeichnet den „Minuspol" (GND = Ground, also Masse), MRESET ist das Master Reset Signal, INTx sind die Interruptletungen, SCL die Takt- und SDA die Datenleitung vom I²C Bus.

Alles andere was Sie evtl. noch brauchen sollten, müssen Sie selbst an die USRBUS Anschlüsse löten.

Wichtiger Hinweis: **Belasten Sie die Versorgungsleitungen VDD und +UB der Erweiterungsanschlüsse nicht mehr als bis jeweils maximal 1A (gilt für beide Pins ZUSAMMEN! Also jeweils die Pins 4+6 (+UB) und 3+5 (VDD))!**

3. Inbetriebnahme

Bevor Sie mit dem RP6 oder seinem Zubehör loslegen, folgen nun einige Sicherheitshinweise die Sie *vor allem* dann beachten sollten, wenn Kinder den RP6 betreiben werden!

Lesen Sie den folgenden Abschnitt daher bitte besonders aufmerksam durch!

3.1. Sicherheitshinweise

Bedingt durch die offene Bauform gibt es beim RP6 einige **spitze Ecken und scharfe Kanten.** Er darf daher ***nicht*** als Spielzeug für Kinder unter 8 Jahren eingesetzt werden. Beaufsichtigen Sie kleine Kinder, die sich beim Betrieb des Roboters im Raum befinden und weisen Sie Ihre Kinder auch auf die hier aufgeführten Gefahren hin!
Betreiben Sie den Roboter nicht, wenn sich freilaufende Kleintiere wie z.B. Hamster im Raum befinden, ansonsten besteht Verletzungsgefahr für die Kleintiere. (*Umgekehrt* könnte der Roboter aber natürlich auch von einem größeren Tier, wie einem Hund oder einer Katze beschädigt werden...)

Bei dem Raupenantrieb gibt es zwischen Rad und Raupenband Einzugsstellen. Diese Bereiche sind beim RP6 weitgehend durch die Radkästen zwischen den Rädern ausgefüllt und dadurch gesichert. Achten Sie beim Betrieb trotzdem darauf, dass Sie **nicht mit den Fingern zwischen Rad und Raupenband** geraten. Die Motoren sind ziemlich kräftig und Sie könnten sich leicht verletzen! Auch zwischen Platine und Raupenband müssen Sie sehr vorsichtig sein!

Achtung: Die Motoren können bereits mit der standard Software die Motorleistung automatisch erhöhen! Je nach Programmierung des Roboters können die Motoren unerwartet anlaufen und es können unvorhergesehene Fahr- und Lenkbewegungen auftreten! **Betreiben Sie den Roboter niemals unbeaufsichtigt!**

3.1.1. Statische Entladungen und Kurzschlüsse

Auf der Oberfläche der Hauptplatine, *dem USB Interface und allen Erweiterungsmodulen* befinden sich **viele nicht abgedeckte Bauteile und Leiterbahnen**. Verursachen Sie keine Kurzschlüsse durch versehentlich abgelegte Metallgegenstände oder Werkzeug!

Die Betriebsspannung ist sehr niedrig und damit für den Menschen ungefährlich. **Viele der Bauelemente sind allerdings elektrostatisch gefährdet und daher sollten Sie die Komponenten auf den Platinen NICHT berühren** wenn es sich vermeiden lässt! Besonders bei trockener Luft (und vor allem wenn man synthetische Kleidung, o.ä. trägt) kann sich der menschliche Körper elektrostatisch aufladen. Das gilt auch für den Roboter selbst - hier ist die Beschaffenheit des Bodenbelags wesentlich. Beim Kontakt mit leitenden Gegenständen baut sich diese Ladung mit einem kleinen Funken ab. Solche Entladungen beim Berühren elektronischer Bauelemente können diese zerstören. Vor dem Hantieren mit dem Roboter oder seinem Zubehör sollten Sie einen großen, geerdeten Gegenstand berühren (z.B. ein PC-Metallgehäuse, eine Wasserleitung oder ein Heizungsrohr), um eventuelle Aufladungen abzubauen. Eine Entladung des Roboters selbst gegen geerdete Gegenstände ist ungefährlich, kann jedoch zu

Programmabstürzen oder unkontrollierter Funktion des Roboters führen.

Alle elektrischen Verbindungen von und zum Gerät sind stets vor Anschluss der Versorgungsspannung herzustellen.
Das Aufstecken oder Abziehen von Verbindungskabeln/Modulen oder das Herstellen oder Lösen von Verbindungen können sonst zur Zerstörung von Teilen des Roboters oder seinem Zubehör führen.

3.1.2. Umgebung des Roboters

Betreiben Sie den Roboter nicht auf Tischflächen oder Bereichen, auf denen Absturzgefahr besteht. Denken Sie hierbei auch an die Kletterfähigkeit des Raupenantriebs! Kleine Hindernisse kann der Roboter problemlos überqueren und leichte Gegenstände wegschieben! Vor Inbetriebnahme des Roboters müssen alle im Erfassungsbereich befindlichen Flüssigkeitsbehälter wie z.B. Kaffeetassen, Flaschen oder Blumenvasen gesichert oder entfernt werden.

Die geschlossene Bodenwanne schützt die Mechanik zwar recht gut gegen grobe Umwelteinflüsse, sie ist aber weder wasser- noch staubdicht. Auch die Elektronik ist weitestgehend ungeschützt. Setzen Sie den Roboter daher nur im trockenen und sauberen Innenbereich ein. Schmutz, Fremdkörper und Feuchtigkeit können die Mechanik und Elektronik beschädigen oder zerstören. Die Umgebungstemperatur darf während des Betriebs 0°C und 40°C nicht unter- bzw. überschreiten.

Es können vor allem innerhalb der Elektromotoren während des Betriebs kleine Funken entstehen. Betreiben Sie den Roboter daher ***auf gar keinen Fall*** in einer Umgebung mit brennbaren oder explosionsgefährdeten Flüssigkeiten, Gasen oder Stäuben!

Bei längerer Nichtverwendung sollte der Roboter nicht in Räumen mit hoher Luftfeuchtigkeit gelagert werden! Weiterhin müssen Sie die Akkus entnehmen, da sie sonst nach einiger Zeit auslaufen könnten!

3.1.3. Versorgungsspannung

Zur Versorgung des Roboters ist eine Gleichspannung von 7.2V vorgesehen, die mit 6 NiMH Akkus erzeugt wird. Die maximale Betriebsspannung beträgt 10V und sollte niemals überschritten werden. Verwenden Sie zum Laden der Akkus nur auf Einhaltung der geltenden Sicherheitsvorschriften geprüfte Ladegeräte!
Behelfsweise kann der Roboter auch mit 6 hochwertigen Alkali-Mangan Batterien betrieben werden. Da normale Batterien schnell leer sein werden und damit hohe Kosten und Umweltbelastung verursachen, sollten Sie aber besser Akkus verwenden! Akkus haben außerdem eine höhere Strombelastbarkeit und können auch bequem im Roboter geladen werden!
Bitte beachten Sie auch die Sicherheits- und Entsorgungshinweise für Batterien und Akkus im Anhang!

Modifikationen am Roboter sollten Sie nur dann selbst durchführen, wenn Sie genau wissen was Sie tun. Sie könnten damit den Roboter irreversibel beschädigen oder sogar sich selbst und andere damit in Gefahr bringen (Es sei nur Zimmerbrand durch überlastete Bauelemente als Beispiel genannt ...)!

3.2. Software Installation

Als nächstes kommen wir zur Software Installation. Die korrekt installierte Software wird für alle nachfolgenden Kapitel unbedingt benötigt.

Es sind Administrator-Rechte erforderlich, also melden Sie sich ggf. vorher als Administrator an Ihrem System an!

Wir empfehlen Ihnen erstmal das gesamte Kapitel in Ruhe durchzulesen und erst dann Schritt für Schritt die Installationsanleitung durchzugehen!

Grundlegende Kenntnis der Bedienung von Computern mit Windows oder Linux Betriebssystem und den gängigen Programmen wie Dateimanager, Webbrowser, Texteditor, Packer (WinZip, WinRAR, unzip o.ä.) und ggf. Linux-Shell etc. **muss vorrausgesetzt werden!** Wenn Sie sich also nur wenig mit Computern auskennen, sollten Sie sich auf jeden Fall gut damit vertraut machen *bevor* Sie den RP6 in Betrieb nehmen! Eine Einführung in die Bedienung von Computern ist nicht Ziel dieser Anleitung und würde den Rahmen bei weitem sprengen! Hier geht es nur um den RP6, dessen Programmierung und die speziell dafür benötigte Software.

3.2.1. Die RP6 CD-ROM

Sie haben vermutlich die RP6 CD-ROM im Laufwerk Ihres Computers – falls doch nicht, legen Sie diese nun bitte ein! Es sollte unter Windows kurz darauf per Autostart das CD Menü erscheinen. Andernfalls können Sie über einen Dateimanager die Datei "start.htm" im Hauptverzeichnis der CD mit einem Webbrowser wie z.B. Firefox öffnen. Die Installationsdateien für Firefox finden Sie übrigens auf der CD im Ordner

```
<CD-ROM-Laufwerk>:\Software\Firefox
```

sofern Sie noch keinen aktuellen Webbrowser installiert haben sollten. (Es sollte mindestens Firefox 1.x oder der Internet Explorer 6 sein...)

Nach Auswahl der Sprache finden Sie im CD Menü neben dieser Anleitung (die es auch zum Download auf unserer Homepage gibt), vielen Informationen, Datenblättern und Fotos auch den Menüpunkt "Software". Hier sind alle Software Tools, der USB Treiber und die Beispielprogramme mit Quellcode für den RP6 zu finden.

Abhängig von den Sicherheitseinstellungen Ihres Webbrowsers können Sie die Installationsprogramme direkt von der CD starten! Wenn Ihr Browser dies aufgrund der Sicherheitseinstellungen nicht erlaubt, müssen Sie die Dateien zunächst in ein Verzeichnis auf Ihrer Festplatte speichern und dann von dort starten. Genaueres dazu steht auf der Software Seite des CD Menüs. Alternativ können Sie natürlich auch direkt in einem Dateimanager auf das CD-Laufwerk wechseln und die Software von dort installieren. Die Verzeichnisnamen sind so gewählt, dass sie eindeutig den entsprechenden Softwarepaketen und Betriebssystemen zugeordnet werden können.

3.2.2. WinAVR - für Windows

Als erstes werden wir WinAVR installieren. *Win*AVR ist aber - wie der Name schon andeutet - nur für *Win*dows verfügbar!

Linux Anwender müssen beim nächsten Abschnitt weiterlesen.

WinAVR (das wird wie das englische Wort "whenever" ausgesprochen) ist eine Samm-

lung von vielen nützlichen und notwendigen Programmen für die Software Entwicklung für AVR Mikrocontroller in der Sprache C. WinAVR enthält neben dem GCC für AVR (das nennt sich dann insgesamt "AVR-GCC", mehr Infos dazu folgen später) auch den komfortablen Quelltexteditor "Programmers Notepad 2", den wir auch für die Programmentwicklung für den RP6 einsetzen werden! WinAVR ist ein privat organisiertes Projekt, hinter dem keine Firma o.ä. steht - es ist kostenlos im Internet verfügbar. Neuere Versionen und weitere Informationen finden Sie hier:

`http://winavr.sourceforge.net/`

Inzwischen wird das Projekt aber auch offiziell von ATMEL unterstützt, und der AVR-GCC lässt sich in AVRStudio, die Entwicklungsumgebung für AVRs von ATMEL, einbinden. Das werden wir in diesem Handbuch aber nicht beschreiben, für unsere Zwecke ist Programmers Notepad besser geeignet.

Die WinAVR Installationsdatei finden Sie auf der CD im Ordner:

`<CD-ROM-Laufwerk>:\Software\AVR-GCC\Windows\WinAVR\`

Die Installation von WinAVR ist sehr einfach und selbsterklärend - normalerweise brauchen keinerlei Einstellungen geändert werden – also einfach immer auf "Weiter" klicken!

Wenn Sie Windows Vista nutzen, müssen Sie auf jeden Fall die neueste Version 20070525 von WinAVR verwenden! Auch mit Windows 2k und XP sollte es problemlos klappen. Falls nicht, können Sie eine der beiden älteren Versionen ausprobieren, die ebenfalls auf der CD zu finden sind (vor Neuinstallation immer bereits installierte WinAVR Versionen wieder deinstallieren!). Offiziell wird Win x64 noch nicht unterstüzt, aber auf der CD findet sich ein Patch für Win x64 Systeme falls es probleme geben sollte. Mehr Infos dazu finden Sie auf der Software Seite des CD Menüs!

3.2.3. AVR-GCC, avr-libc und avr-binutils - für Linux

Windows Anwender können diesen Abschnitt überspringen!

Unter Linux kann es schon ein wenig aufwändiger werden. Bei einigen Distributionen sind die benötigten Pakete zwar schon vorhanden, aber meist nur veraltete Versionen.

Deshalb müssen Sie neuere Versionen kompilieren und einrichten.

Wir können hier nicht im Detail auf jede der zahlreichen Linux Distributionen wie SuSE, Ubuntu, RedHat/Fedora, Debian, Gentoo, Slackware, Mandriva etc. pp. in zig verschiedenen Versionen mit ihren jeweiligen Eigenheiten eingehen und beschreiben das daher nur allgemein.

Das gilt auch für alle anderen Linux Abschnitte in diesem Kapitel!

Das hier beschriebene Vorgehen muss also bei Ihnen nicht unbedingt zum Erfolg führen. Oft kann es hilfreich sein, wenn Sie im Internet z.B. nach "<LinuxDistribution> avr gcc" o.ä. suchen (Verschiedene Schreibweisen ausprobieren). Auch das gilt für alle anderen Linux Abschnitte - natürlich mit angepassen Suchbegriffen! Falls Sie Probleme bei der Installation des AVR-GCC haben, können Sie auch mal in unserem oder im Roboternetz Forum nachschauen bzw. in einem der zahlreichen Linux Foren.

Zunächst müssen Sie evtl. schon installierte Versionen des avr-gcc, der avr-binutils und der avr-libc deinstallieren – wie schon gesagt sind diese meist veraltet. Das können Sie mit dem jeweiligen Paketmanager ihrer Distribution tun indem Sie nach „avr"

suchen und die drei oben genannten Pakete deinstallieren – sofern diese überhaupt vorhanden sind.

Ob der avr-gcc schon installiert ist oder nicht und wenn ja wo, können Sie über eine Konsole z.B. leicht mit

```
> which avr-gcc
```

herausfinden. Sollte hier ein Pfad angezeigt werden, ist schon eine Version installiert.

Geben Sie in diesem Fall einfach mal:

```
> avr-gcc --version
```

ein und schauen Sie sich die Ausgabe an. Sollte eine Versionsnummer kleiner 3.4.6 angezeigt werden, müssen Sie diese alte Version auf jeden Fall deinstallieren. Wenn die Versionsnummer zwischen 3.4.6 und 4.1.0 liegt, können Sie erstmal versuchen ob Sie Programme kompilieren können (s. nachfolgende Kapitel) und erst wenn das fehlschlägt, die neuen Tools installieren. Wir installieren im Folgenden die derzeit aktuelle Version 4.1.1 (Stand von März 2007) mit einigen wichtigen Patches.

Werden die oben genannten Pakete nicht im Paketmanager angezeigt, obwohl definitiv schon ein avr-gcc vorhanden ist, müssen Sie die entsprechenden Binärdateien manuell löschen – also die /bin, /usr/bin usw. Verzeichnisse nach allen Dateien die mit „avr-“ anfangen absuchen und diese dann löschen (natürlich NUR diese Dateien und sonst nichts anderes!). Eventuell vorhandene Verzeichnisse wie /usr/avr oder /usr/local/avr müssen ebenfalls gelöscht werden.

Achtung: Sie müssen unbedingt sicherstellen, dass die normalen Linux Entwicklungstools wie GCC, make, binutils, libc, etc. installiert sind bevor Sie mit dem Übersetzen und der Installation beginnen können! Das tun Sie am besten über den Paketmanager Ihrer Distribution. Jede Linux Distribution sollte die benötigten Pakete schon auf der Installations CD mitliefern bzw. aktuelle Pakete über das Internet bereitstellen.

Stellen Sie sicher, dass das Programm „texinfo“ installiert ist. Installieren Sie bitte ggf. das entsprechende Paket bevor Sie weitermachen – sonst klappt es nicht!

Ist das erledigt, kann mit der eigentlichen Installation begonnen werden.

Es gibt nun zwei Möglichkeiten, entweder man macht alles von Hand, oder man nutzt ein sehr einfach anzuwendendes Installationsskript.

Wir empfehlen es zunächst mit dem Skript zu versuchen, wenn das nicht klappt kann man immer noch den Compiler von Hand einrichten!

Achtung: Sie sollten für die Installation noch genug freien Speicherplatz auf der Festplatte zur Verfügung haben! Temporär werden mehr als 400MB benötigt. Über 300MB davon können nach der Installation wieder gelöscht werden, aber während der Übersetzung braucht man den Platz.

Viele der nachfolgenden Installationsschritte erfordern **ROOT RECHTE,** also loggen Sie sich ggf. mit „su“ als root ein oder führen Sie die kritischen Befehle mit „sudo“ o.ä. aus, wie man es z.B. bei Ubuntu machen muss (das Installationsskript, mkdir in /usr/local Verzeichnissen und make install brauchen root Rechte).

Achten Sie im Folgenden bitte auf EXAKTE Schreibweise aller Befehle! Jedes Zeichen ist wichtig und auch wenn einige Befehle evtl. etwas seltsam aussehen – das ist alles richtig so und kein Tippfehler! (`<CD-ROM-Laufwerk>` muss man natürlich trotzdem mit dem Pfad des CD-ROM-Laufwerks ersetzen!)

Alle für uns relevanten Installationsdateien für den avr-gcc, avr-libc und binutils finden Sie auf der CD im Ordner:

```
<CD-ROM-Laufwerk>:\Software\avr-gcc\Linux
```

Zunächst müssen Sie alle Installationsdateien in ein Verzeichnis auf Ihrer Festplatte kopieren – **das gilt für beide Installationsvarianten!** Hier nutzen wir das Home Verzeichnis (übliche Abkürzung für das aktuelle Home Verzeichnis ist die Tilde: „~"):

```
> mkdir ~/RP6
> cd <CD-ROM-Laufwerk>/Software/avr-gcc/Linux
> cp * ~/RP6
```

Die Dateien können Sie nach der erfolgreichen Installation natürlich wieder löschen um Platz zu sparen!

3.2.3.1. Automatisches Installationsskript

Wenn man das Skript mit chmod ausführbar gemacht hat, kann es sofort losgehen:

```
> cd ~/RP6
> chmod -x avrgcc_build_and_install.sh
> ./avrgcc_build_and_install.sh
```

Die Nachfrage ob man mit dieser Konfiguration installieren möchte oder nicht, können Sie mit „y" beantworten.

ACHTUNG: Das Übersetzen und Installieren wird dann je nach Rechenleistung Ihres Systems einige Zeit in Anspruch nehmen. (z.B. etwa 15 min auf einem 2GHz CoreDuo Notebook – bei langsameren Systemen evtl. entsprechend länger)

Das Skript spielt auch einige Patches ein – das sind diese ganzen .diff Dateien die in dem Verzeichnis liegen.

Wenn alles klappt, sollte ganz zum Schluss folgendes erscheinen:

```
(./avrgcc_build_and_install.sh)
(./avrgcc_build_and_install.sh) installation of avr GNU tools complete
(./avrgcc_build_and_install.sh) add /usr/local/avr/bin to your path to use the avr GNU tools
(./avrgcc_build_and_install.sh) you might want to run the following to save disk space:
(./avrgcc_build_and_install.sh)
(./avrgcc_build_and_install.sh) rm -rf /usr/local/avr/source /usr/local/avr/build
```

Dann können Sie wie es dort vorgeschlagen wird

```
rm -rf /usr/local/avr/source /usr/local/avr/build
```

ausführen! Das löscht alle temporären Dateien die Sie normalerweise nicht mehr benötigen.

Jetzt können Sie den nächsten Abschnitt überspringen und noch den Pfad auf die avr tools setzen.

Sollte die Ausführung des Skriptes fehlschlagen, müssen Sie sich genau die Fehlermeldungen ansehen (auch mal in der Konsole etwas hochscrollen) – meist fehlen dann irgendwelche Programme die man vorher noch installieren muss (wie z.B. das oben erwähnte texinfo). Bevor Sie nach einem Fehler weitermachen, sollten Sie die bereits erzeugten Dateien im standard Installationsverzeichnis „/usr/local/avr" vorsichtshalber löschen – am besten das ganze Verzeichnis.

Wenn Sie nicht wissen, was da genau falsch gelaufen ist, bitte alle Kommandozeilenausgaben in einer Datei speichern und damit an den Support wenden. Bitte immer so viele Informationen wie möglich mitsenden! Dann wird es einfacher, Ihnen zu helfen.

3.2.3.2. Manuelle Installation

Wenn Sie den Compiler lieber von Hand einrichten wollen oder die Installation mit dem Skript nicht klappt, können Sie nach den Anweisungen im folgenden Abschnitt vorgehen.

Die Beschreibung hier orientiert sich an diesem Artikel:

`http://www.nongnu.org/avr-libc/user-manual/install_tools.html`

der auch in der AVR Libc Dokumentation im PDF Format auf der CD zu finden ist:

`<CD-ROM-Laufwerk>:\Software\Documentation\avr-libc-user-manual-1.4.5.pdf`

ab PDF Seite 240 (bzw. 232 nach der Seitenzahl des Dokuments).

Wir fassen uns hier zwar sehr viel kürzer, spielen aber gleich noch ein paar wichtige Patches ein – ohne diese funktionieren einige Dinge nicht richtig.

Zunächst müssen wir uns ein Verzeichnis erstellen, in das wir alle Tools installieren werden. Das sollte `/usr/local/avr` sein.
Also in einer Konsole **ALS ROOT** folgendes eingeben:

```
> mkdir /usr/local/avr
> mkdir /usr/local/avr/bin
```

Es muss nicht unbedingt dieses Verzeichnis sein. Wir legen dafür einfach die Variable `$PREFIX` für dieses Verzeichnis an:

```
> PREFIX=/usr/local/avr
> export PREFIX
```

Das muss nun noch unbedingt der PATH Variable hinzugefügt werden:

```
> PATH=$PATH:$PREFIX/bin
> export PATH
```

Binutils für AVR

Nun müssen Sie den Quellcode der Binutils entpacken und ein paar Patches einspielen. Wir nehmen hier an, dass Sie alles ins Home Verzeichnis ~/RP6 kopiert haben:

```
> cd ~/RP6
> bunzip2 -c binutils-2.17.tar.bz2 | tar xf -
> cd binutils-2.17
> patch -p0 < ../binutils-patch-aa.diff
> patch -p0 < ../binutils-patch-atmega256x.diff
> patch -p0 < ../binutils-patch-coff-avr.diff
> patch -p0 < ../binutils-patch-newdevices.diff
> patch -p0 < ../binutils-patch-avr-size.diff
> mkdir obj-avr
> cd obj-avr
```

Nun wird das configure Skript ausgeführt:

```
> ../configure --prefix=$PREFIX --target=avr --disable-nls
```

Dieses Skript ermittelt, was auf Ihrem System verfügbar ist und erzeugt dementsprechend passende Makefiles. Jetzt können die Binutils übersetzt und installiert werden:

```
> make
> make install
```

Das kann je nach Rechenleistung Ihres Systems schon ein paar Minuten dauern – das gilt auch für die beiden nächsten Abschnitte – vor allem für den GCC!

GCC für den AVR

Der GCC wird ähnlich wie die Binutils gepatcht, übersetzt und installiert:

```
> cd ~/RP6
> bunzip2 -c gcc-4.1.1.tar.bz2 | tar xf -
> cd gcc-4.1.1
> patch -p0 < ../gcc-patch-0b-constants.diff
> patch -p0 < ../gcc-patch-attribute_alias.diff
> patch -p0 < ../gcc-patch-bug25672.diff
> patch -p0 < ../gcc-patch-dwarf.diff
> patch -p0 < ../gcc-patch-libiberty-Makefile.in.diff
> patch -p0 < ../gcc-patch-newdevices.diff
> patch -p0 < ../gcc-patch-zz-atmega256x.diff
> mkdir obj-avr
> cd obj-avr
> ../configure --prefix=$PREFIX --target=avr --enable-languages=c,c++ \
     --disable-nls --disable-libssp –with-dwarf2
> make
> make install
```

Nach dem \ kann man einfach Enter drücken und weiterschreiben – so kann der Befehl auf mehrere Zeilen aufgeteilt werden. Kann man aber auch ganz weglassen.

AVR Libc

Und schließlich noch die AVR libc:

```
> cd ~/RP6
> bunzip2 -c avr-libc-1.4.5.tar.bz2 | tar xf -
> cd avr-libc-1.4.5
> ./configure --prefix=$PREFIX --build=`./config.guess` --host=avr
> make
> make install
```

Achtung: bei –build=`./config.guess` darauf achten auch den „Accent grave" (à <-- den Strich da auf dem a! Neben der Backspace Taste – rechts oben auf der Tastatur, einmal mit Shift diese Taste drücken und danach die Leertaste) und kein normales Hochkomma oder Anführungszeichen zu benutzen, sonst klappt es nicht.

3.2.3.3. Pfad setzen

Sie müssen jetzt dafür sorgen, dass das Verzeichnis `/usr/local/avr/bin` auch in der Pfad Variablen eingetragen ist – sonst kann man den avr-gcc nicht aus der Konsole bzw. aus Makefiles heraus aufrufen. Dazu müssen Sie den Pfad in die Datei `/etc/profile` bzw. `/etc/environment` o.ä. (variiert von Distribution zu Distribution) eintragen – mit einem Doppelpunkt „:" getrennt von den anderen schon vorhandenen Einträgen. In der Datei könnte das dann *in etwa* so aussehen:

```
PATH="/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/usr/local/avr/bin"
```

Jetzt in einer beliebigen Konsole „avr-gcc --version" eingeben wie weiter oben beschrieben – wenn das funktioniert, ist die Installation gelungen!

3.2.4. Java 6

Der RP6Loader (Infos dazu s.u.) wurde für die Java Plattform entwickelt und ist unter Windows und Linux verwendbar (theoretisch auch andere Betriebsysteme wie OS X, aber hier kann AREXX Engineering leider noch keinen offiziellen Support leisten). Damit das funktioniert, ist es notwendig, ein aktuelles Java Runtime Environment (JRE) zu installieren. Oft haben Sie dies bereits auf dem Rechner, allerdings muss es mindestens Version 1.6 (= Java 6) sein! Falls Sie also noch kein JRE oder JDK installiert haben, müssen Sie zunächst das auf der CD mitgelieferte JRE 1.6 der Firma SUN Microsystems installieren, oder alternativ eine neuere Version von `http://www.java.com` oder `http://java.sun.com` downloaden.

3.2.4.1. Windows

Das JRE 1.6 befindet sich für Windows in folgendem Ordner:

```
<CD-ROM-Laufwerk>:\Software\Java\JRE6\Windows\
```

Unter Windows ist die Installation von Java sehr einfach - Sie müssen nur den Setup starten und den Anweisungen auf dem Bildschirm folgen - fertig. Den nächsten Abschnitt können Sie überspringen.

3.2.4.2. Linux

Unter Linux ist die Installation meistens auch relativ problemlos möglich, bei einigen Distributionen kann es aber ein wenig Handarbeit erfordern.

In diesem Ordner:

```
<CD-ROM-Laufwerk>:\Software\Java\JRE6\
```

finden Sie das JRE1.6 als RPM (SuSE, RedHat etc.) und als selbstextrahierendes Archiv „.bin“. Unter Linux ist es besser wenn Sie zunächst im Paketmanager Ihrer jeweiligen Distribution nach Java Paketen suchen (Suchbegriffe z.B. „java“, „sun“, „jre“, „java6“ ...) und dann diese Distributionseigenen Pakete verwenden und <u>nicht</u> die auf dieser CD-ROM! Achten Sie aber unbedingt darauf Java 6 (= 1.6) oder ggf. eine neuere Version zu installieren und keine ältere Version!

Unter Ubuntu oder Debian funktioniert das RPM Archiv nicht direkt – hier müssen Sie die Paketmanager Ihrer jeweiligen Distribution bemühen um an ein passendes Installationspaket zu kommen. Das RPM sollte bei vielen anderen Distributionen wie RedHat/Fedora und SuSE aber problemlos funktionieren. Falls nicht, bleibt noch der Weg das JRE aus dem selbstextrahierenden Archiv (.bin) zu entpacken (z.B. nach /usr/lib/Java6) und dann manuell die Pfade zum JRE zu setzen (PATH und JAVA_HOME etc.).

Bitte beachten Sie hier auch die Installationsanweisungen von Sun – die ebenfalls im oben genannten Verzeichnis und auf der Java Website (s.o.) zu finden sind!

Ob Java korrekt installiert wurde, können Sie in einer Konsole überprüfen indem Sie den Befehl „`java -version`“ ausführen. Es sollte in etwa folgende Ausgabe erscheinen:

```
java version "1.6.0"
Java(TM) SE Runtime Environment (build 1.6.0-b105)
Java HotSpot(TM) Client VM (build 1.6.0-b105, mixed mode, sharing)
```

Steht dort etwas ganz anderes, haben Sie entweder die falsche Version installiert, oder auf Ihrem System ist noch eine andere Java VM installiert.

3.2.5. RP6Loader

Der RP6Loader wurde entwickelt, um komfortabel neue Programme in den RP6 und alle Erweiterungsmodule laden zu können (sofern diese über einen Mikrocontroller mit kompatiblem Bootloader verfügen). Weiterhin sind ein paar nützliche Zusatzfunktionen integriert, wie z.B. ein einfaches Terminalprogramm.

Den RP6Loader selbst braucht man nicht zu installieren – das Programm kann einfach irgendwo in einen neuen Ordner auf die Festplatte kopiert werden. Der RP6Loader befindet sich in einem Zip-Archiv auf der CD-ROM:

```
<CD-ROM-Laufwerk>:\Software\RP6Loader\RP6Loader.zip
```

Dieses müssen Sie nur irgendwo auf die Festplatte entpacken – z.B. in einen neuen Ordner `C:\Programme\RP6Loader` (o.ä.). In diesem Ordner finden Sie dann die Datei RP6Loader.exe und können sie mit einem Doppelklick starten.

Das eigentliche RP6Loader Programm liegt im Java Archive (JAR) RP6Loader_lib.jar. Dieses können Sie alternativ auch von der Kommandozeile aus starten.

Unter Windows:

```
java -Djava.library.path=".\lib" -jar RP6Loader_lib.jar
```

Linux:

```
java -Djava.library.path="./lib" -jar RP6Loader_lib.jar
```

Diese lange -D Option ist notwendig, damit die JVM auch alle verwendeten Bibliotheken finden kann. Unter Windows braucht man das aber nicht und kann einfach die .exe Datei zum Starten verwenden und für Linux gibt es ein Shell Skript „RP6Loader.sh". Das Skript muss evtl. zunächst noch ausführbar gemacht werden (`chmod -x ./RP6Loader.sh`). Danach kann man es in einer Konsole mit „./RP6Loader.sh" starten.

Es empfiehlt sich eine Verknüpfung auf dem Desktop oder im Startmenü anzulegen, um den RP6Loader bequem starten zu können. Unter Windows geht das z.B. einfach indem man rechts auf die Datei RP6Loader.exe klickt und dann im Menü „Senden an" auf „Desktop (Verknüpfung erstellen)" klickt.

3.2.6. RP6 Library, RP6 CONTROL Library und Beispielprogramme

Die RP6Library und die zugehörigen Beispielprogramme befinden sich in einem Zip-Archiv auf der CD:

```
<CD-ROM-Laufwerk>:\Software\RP6Examples\RP6Examples.zip
```

Sie können diese einfach direkt in ein Verzeichnis Ihrer Wahl auf die Festplatte entpacken. Am besten entpacken Sie die Beispielprogramme in einen Ordner auf einer Daten Partition. Oder in den „Eigene Dateien" Ordner in einem Unterordner „RP6\Examples\" bzw. unter Linux ins Home Verzeichnis. Das steht Ihnen aber völlig frei.

Die einzelnen Beispielprogramme werden noch später im Softwarekapitel besprochen!

In dem Archiv sind auch schon die Beispiele für das RP6 CONTROL M32 Erweiterungsmodul und die entsprechenden Library Dateien enthalten!

3.3. Anschluss des USB Interfaces – Windows

Linux Anwender können beim nächsten Abschnitt weiterlesen!

Zur Installation des USB Interfaces gibt es mehrere Möglichkeiten. Die einfachste Möglichkeit ist es, **den Treiber VOR dem ersten Anschließen des Geräts zu installieren.** Auf der CD befindet sich ein Installationsprogramm für den Treiber.

Für **32 und 64 Bit Windows XP, Vista, Server 2003 und 2000** Systeme:

```
<CD-ROM-Laufwerk>:\Software\USB_DRIVER\Win2k_XP\CDM_Setup.exe
```

Für alte **Win98SE/Me** Systeme *gibt es so ein komfortables Programm leider nicht – hier muss ein älterer Treiber von Hand installiert werden nachdem man das Gerät angeschlossen hat (s.u.).*

Das CDM Installationsprogramm müssen Sie einfach nur ausführen – es gibt nur eine kurze Rückmeldung, sobald der Treiber installiert wurde, sonst passiert nichts weiter.

Dann können Sie das USB Interface an den PC anschließen. *BITTE NOCH NICHT MIT DEM ROBOTER VERBINDEN!* Einfach nur über das USB Kabel mit dem PC verbinden! Dabei sollten Sie darauf achten, die Platine des USB Interfaces nur am Rand oder am USB Stecker bzw. an der Kunststoffwanne des Programmiersteckers anzufassen (s. Sicherheitshinweise zu statischen Entladungen)! Sie sollten besser *keine* der Bauteile auf der Platine, Lötstellen oder die Kontakte des Wannensteckers berühren wenn nicht unbedingt nötig, um statische Entladungen zu vermeiden!

Der zuvor installierte Treiber wird nun automatisch für das Gerät verwendet ohne das Sie noch etwas tun brauchen. Es erscheinen bei Windows XP/2k kleine Sprechblasen unten über der Taskleiste – die letzte Meldung sollte in etwa „Die Hardware wurde installiert und kann nun verwendet werden!" lauten!

Wenn Sie das USB Interface doch schon vor der Installation angeschlossen haben (oder Win98/Me benutzen) – auch nicht schlimm. Dann werden Sie von Windows nach einem Treiber gefragt. Auch diese Installationsvariante ist möglich, der Treiber befindet sich auch in entpackter Form auf der CD!

Wenn dies bei Ihnen der Fall ist, erscheint (unter Windows) für gewöhnlich ein Dialog zum Installieren eines neuen Gerätetreibers. Sie müssen dem System dann den Pfad angeben unter dem es den Treiber finden kann. Bei Windows 2k/XP muss man erst auswählen den Treiber manuell zu installieren und natürlich keinen Webdienst o.ä. zu suchen. Der Treiber befindet sich in unserem Fall auf der CD in den oben genannten Verzeichnissen.

Also einfach das jeweilige Verzeichnis für Ihre Windows Version angeben und evtl. noch ein paar Dateien die das System nicht selbstständig findet (sind alle in den weiter unten genannten Verzeichnissen!)...

Bei Windows XP oder späteren Versionen folgt oft (hier normalerweise nicht, da die FTDI Treiber signiert sind) noch ein Hinweis das der Treiber nicht von Microsoft signiert/verifiziert worden ist - das ist irrelevant und kann bedenkenlos bestätigt werden.

Für **32 und 64 Bit Windows XP, Vista, Server 2003 und 2000** Systeme:

```
<CD-ROM-Laufwerk>:\Software\USB_DRIVER\Win2k_XP\FTDI_CDM2\
```

Für ältere **Windows 98SE/Me** Systeme:

```
<CD-ROM-Laufwerk>:\Software\USB_DRIVER\Win98SE_ME\FTDI_D2XX\
```

Bei einigen älteren Windows Versionen wie Win98SE ist evtl. nach Installation des Treibers ein Neustart erforderlich! **ACHTUNG:** Unter **Win98/Me** funktioniert nur einer von beiden Treibern: Virtual Comport oder der D2XX Treiber von FTDI! Hier gibt es leider keinen Treiber der beide Funktionen integriert und es steht normalerweise kein virtueller Comport zur Verfügung, da der RP6Loader unter Windows standardmäßig die D2XX Treiber verwendet (das kann man auch ändern - kontaktieren Sie hier ggf. unser Support Team!).

3.3.1. Überprüfen ob das Gerät richtig angeschlossen ist

Um zu überprüfen ob das Gerät korrekt installiert worden ist, kann man unter Windows XP, 2003 und 2000 neben dem RP6Loader auch den Gerätemanager verwenden:
Rechtsklick auf den Arbeitsplatz --> Eigenschaften --> Hardware --> Gerätemanager
ODER alternativ: Start --> Einstellungen --> Systemsteuerung --> Leistung und Wartung --> System --> Hardware --> Gerätemanager
und dort in der Baumansicht unter "Anschlüsse (COM und LPT)" nachsehen ob ein "USB-Serial Port (COMX)" zu sehen ist - wobei das X für die Portnummer steht oder unter „USB-Controller" nach einem „USB Serial Converter" suchen!

3.3.2. Treiber später wieder Deinstallieren

Sollten Sie den Treiber jemals wieder deinstallieren wollen *(Nein, das tun Sie jetzt bitte nicht - ist nur ein Hinweis falls Sie das jemals brauchen sollten)*: Wenn Sie das CDM Installationsprogramm verwendet haben, können Sie das direkt über Start-->Einstellungen-->Systemsteuerung-->Software tun. In der dortigen Liste finden Sie einen Eintrag des „FTDI USB Serial Converter Drivers" – diesen auswählen und dort dann auf deinstallieren klicken!

Wenn Sie den Treiber von Hand installiert haben, können Sie das Programm "FTUNIN.exe" im Verzeichnis des jeweiligen USB Treibers für Ihr System ausführen!

Achtung: USB-->RS232 Adapter mit FTDI Chipsatz verwenden meist ebenfalls diesen Treiber!

3.4. Anschluss des USB Interfaces – Linux

Windows Anwender können diesen Abschnitt überspringen!

Bei Linux mit Kernel 2.4.20 oder höher ist der benötigte Treiber schon vorhanden (zumindest für das kompatible Vorgängermodell FT232BM des Chips auf unserem USB Interface, dem FT232R), das Gerät wird automatisch erkannt und Sie brauchen nichts weiter zu tun. Falls es doch mal Probleme gibt, erhalten Sie Linux Treiber (und Support und auch evtl. neuere Treiber) direkt von FTDI:

http://www.ftdichip.com/

Unter Linux kann man nachdem man das Gerät angeschlossen hat mit:

```
cat /proc/tty/driver/usbserial
```

anzeigen lassen ob der USB-Serial Port korrekt installiert worden ist. Mehr braucht man hier normalerweise nicht zu tun.
Allerdings sei noch darauf hingewiesen, dass der RP6Loader unter Windows die D2XX Treiber verwendet und dort die vollständigen USB Bezeichnungen in der Portliste auftauchen (z.B. „USB0 | RP6 USB Interface | serialNumber“). Unter Linux sind es stattdessen die virtuellen Comport Bezeichnungen /dev/ttyUSB0, /dev/ttyUSB1 etc.. Es werden auch die normalen Comports angezeigt, als „dev/ttyS0“ usw.. Hier müssen Sie ausprobieren welcher Port der richtige ist!
Für Linux ist leider kein so einfach zu installierender Treiber verfügbar der beides bereitstellt, von daher war es hier sinnvoller die Virtual Comport Treiber die ohnehin schon im Kernel vorhanden sind zu verwenden. Die D2XX Treiber würden bei der Installation auch noch einiges an Handarbeit erfordern...

Sollten unter Linux keine Ports angezeigt werden, lesen Sie bitte im Anhang A weiter!

3.5. Software Installation abschließen

Das war es auch schon mit der Installation der Software und des USB Interfaces!

Jetzt könnten Sie noch die wichtigsten Dateien von der CD auf die Festplatte kopieren (vor allem den kompletten Ordner „Documentation“ und falls nicht schon geschenen die Beispielprogramme). Dann müssen Sie nicht ständig die CD suchen wenn Sie diese Dateien benötigen! Die Ordner auf der CD sind alle so benannt, dass sie eindeutig den jeweiligen Softwarepaketen bzw. der jew. Dokumentation zugeordnet werden können! Sollten Sie die CD einmal "verlegen", können Sie die wichtigsten Dateien wie dieses Handbuch, den RP6Loader und die Beispielprogramme auch von der AREXX Homepage downloaden. Dort finden Sie auch Links zu den anderen Softwarepaketen, die Sie benötigen.

3.6. Akkus einbauen

Jetzt kommen wir endlich zum Roboter selbst. Dieser benötigt natürlich noch 6 Akkus!

Wir empfehlen unbedingt qualitativ hochwertige NiMH Mignon Akkus (Hersteller z.B. Sanyo, Panasonic, o.ä.) mit einer realen Kapazität von mindestens 2000mAh (optimal sind 2500mAh) zu verwenden! Normale Batterien sollten Sie nicht verwenden. Dies wäre auf Dauer nicht nur sehr teuer, sondern würde auch unnötigerweise die Umwelt belasten.

Die Akkus sollten am besten **vor dem Einbau schon vorgeladen sein!** Achten Sie darauf, **dass alle Akkus den gleichen Ladezustand haben** (also alle frisch aufgeladen oder alle entladen) und möglichst neu sind! Akkus verlieren mit zunehmendem Alter, Zahl der Ladezyklen, Art der Ladung und Umgebungstemperatur mehr und mehr an Kapazität. Also nicht unbedingt die alten Akkus nehmen die schon seit 3 Jahren unbenutzt im Schrank rumgelegen haben...

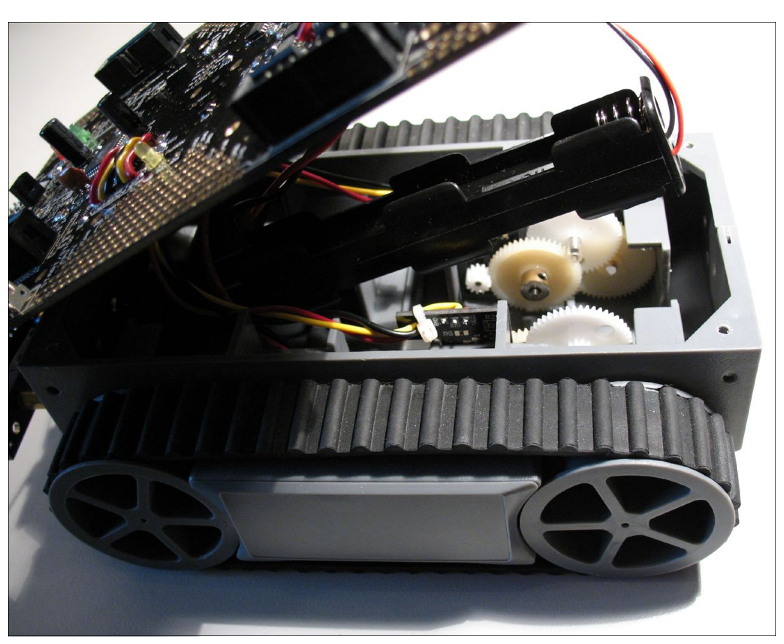
Wenn Sie später ein externes Steckerladegerät verwenden (empfohlen, jedoch <u>nicht</u> im Lieferumfang enthalten!), brauchen Sie die Akkus nur <u>einmalig</u> einzubauen! Wir empfehlen dringend die Verwendung eines Mikrocontroller gesteuerten Ladegeräts um die Akkus schonend zu laden! Verwenden Sie nur zugelassene und geprüfte Ladegeräte!

Wenn Sie noch kein externes Ladegerät mit passendem Anschluss haben, müssen Sie die Akkus in jeden Fall vor dem Einbau laden und auch jedesmal wieder ausbauen, neu aufladen und wieder einbauen wenn diese leer sind!

Zum Einbau der Akkus:

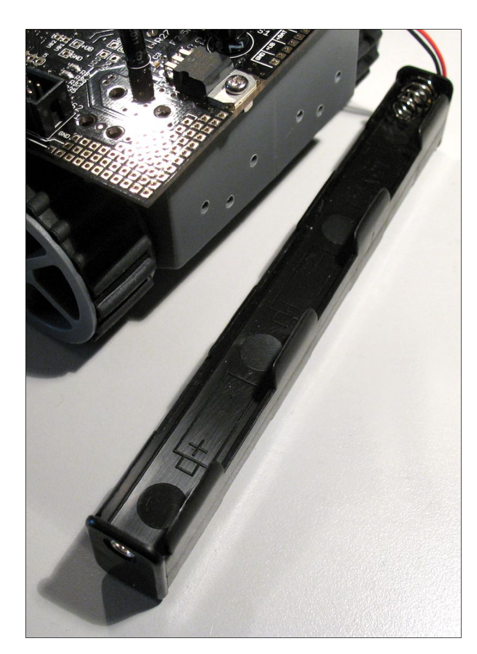
Als erstes müssen Sie die vier Befestigungsschrauben (s. Abb.) des Mainboards lösen!

Heben Sie dann vorsichtig das Mainboard hinten an (a. Abb.).



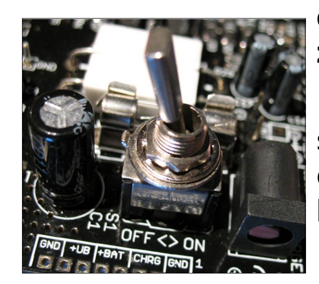
Den kleinen 3 poligen Stecker der Bumper Platine (s. Abb) müssen Sie NICHT unbedingt lösen! Bitte fassen Sie das Mainboard möglichst nur an den Rändern oder an größeren Kunststoffteilen an, um elektrostatische Entladungen zu vermeiden!

Das Mainboard ist mit einigen fest verlöteten Kabeln mit den Motoren, den Drehgebern und dem Akkuhalter verbunden. Schieben Sie diese Kabel - je nachdem wie sie liegen - vorsichtig beiseite.

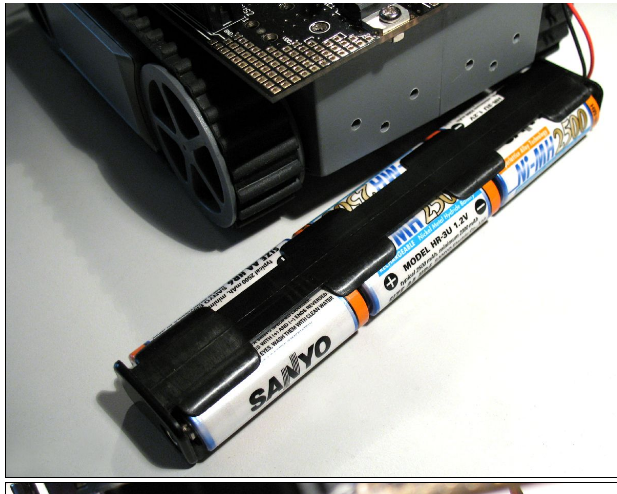


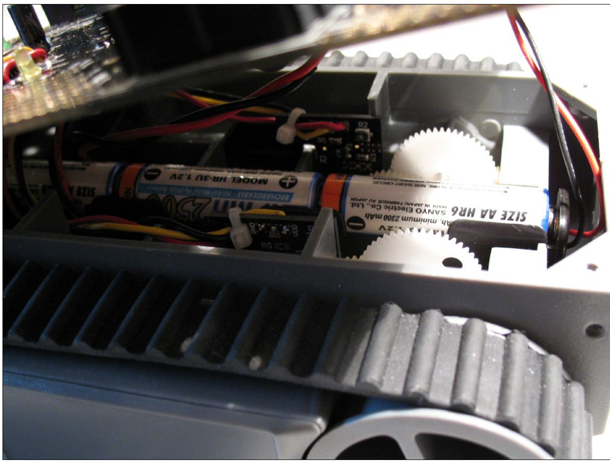
Dann nehmen Sie den schwarzen in der Mitte liegenden Akkuhalter nach hinten heraus (s. Abb.).

Stellen Sie nun bitte unbedingt sicher, dass der Schalter auf dem Mainboard des Roboters in der Position OFF steht, also in Richtung des Textes "OFF", bzw. in Richtung des großen zylinderförmigen Kondensators auf dem Mainboard zeigt (s. Abb)!



Bevor Sie den Roboter wieder einschalten, müssen Sie kontrollieren ob die Akkus richtig herum eingelegt sind.





Jetzt können Sie 6 NiMH Mignon Akkus ***<u>POLUNGSRICHTIG</u>*** in den Akkuhalter einlegen! **Vorsicht: Wenn Sie die Akkus falsch herum einlegen wird die Sicherung normalerweise durchbrennen!**

<u>Es kann aber auch passieren, dass Teile der Elektronik beschädigt werden!</u>

Legen Sie die Akkus also besser gleich richtig herum ein, um derartige Probleme zu vermeiden! **Es sind Markierungen im Akkuhalter und auch auf den Akkus** ((+) und (-), der Minuspol (die flache Seite) der Akkus muss in Richtung der Federn im Akkuhalter zeigen...)!

<u>Kontrollieren Sie lieber dreimal, ob die Akkus auch wirklich richtig herum in den Akkuhalter eingesetzt sind!</u>

Legen Sie nun den Akkuhalter wieder in das Chassis und **achten Sie darauf, dass alle Kabel gut im Chassis verstaut sind und nicht die Zahnräder der beiden Getriebe berühren können!**

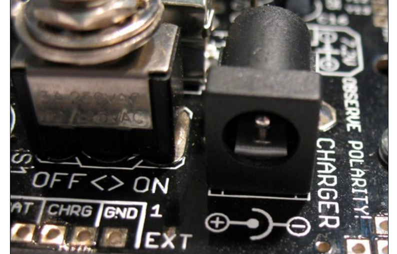
Da der Roboter nun ohnehin schon aufgeschraubt ist, können Sie auch kurz noch die beiden Getriebe und Drehgeber begutachten und schauen, ob beim Transport nichts beschädigt wurde oder sich evtl. gelockert hat. Drehen Sie bitte ***einmal ganz vorsichtig und langsam*** an den Hinterrädern! Eine halbe Umdrehung hin und zurück sollte schon ausreichen. Es sollte zwar ein deutlicher Widerstand zu spüren sein, aber das Rad sollte sich trotzdem gut drehen lassen. Die Zahnräder müssen sich frei drehen können! Bitte Anhang A beachten!

Das Mainboard wird nun wieder auf das Chassis gesetzt. Eventuell zwischen Mainboard und Plastikstege des Chassis geratene Kabel schieben Sie entweder mit einem Finger, oder vorsichtig mit einem langen Schraubendreher beiseite, **damit das Mainboard plan auf dem Chassis aufliegen kann!** Bevor Sie es danach wieder festschrauben, sollten Sie **nochmals überprüfen** ob keine Kabel zwischen Mainboard und Chassis oder in die Getriebe geraten sind! Dann können Sie das Mainboard wieder mit den 4 Schrauben befestigen – Fertig!

3.7. Laden der Akkus

Sofern die Akkus nicht wie empfohlen ohnehin schon geladen sind, müssen Sie diese nun mit einem externen Ladegerät laden. Der Hauptschalter muss dazu auch weiterhin in der Position OFF stehen bleiben! Die Akkus können nur im ausgeschalteten Zustand geladen werden. Der Hauptschalter verbindet die Akkus entweder mit der Elektronik des RP6, oder mit der Ladebuchse.

Achten Sie darauf, das Ladegerät polungsrichtig an die Ladebuchse (mit „Charger" beschriftet) neben dem Hauptschalter des Roboters anzuschließen! Die Polung ist auch auf dem Mainboard vor dem Stecker aufgedruckt (s. Abb).



Der Minuspol liegt AUSSEN, der Pluspol INNEN!

Die Ladezeit ist abhängig vom verwendeten Ladegerät und den Akkus (bei Mikrocontroller gesteuerten Ladegeräten wie dem Voltcraft 1A / 2A Delta Peak Schnellladegerät oder dem Ansmann ACS110 / 410 etwa 3 bis 4, bei normalen wie dem AC48 etwa 14 Stunden) - genaueres dazu entnehmen Sie bitte dem Handbuch Ihres jeweiligen Ladegeräts!

Schalten Sie den Roboter während des Ladervorgangs NICHT ein! Entfernen Sie das Ladegerät bevor Sie den Roboter wieder einschalten!

3.8. Der erste Test

ACHTUNG! Lesen Sie diesen und den folgenden Abschnitt komplett durch bevor Sie den Roboter anschalten! Sollte dann etwas nicht so ablaufen wie hier beschrieben, schalten Sie den Roboter am besten *sofort* aus und notieren Sie sich genau was falsch gelaufen ist! Sofern das Kapitel "Fehlersuche" keine Lösung liefert, wenden Sie sich bitte an den Support!

Jetzt kann es los gehen - der Roboter wird nun das erste Mal eingeschaltet! Schalten Sie den Roboter am Hauptschalter ein. Es sollten nun die zwei mittleren roten Status LEDs aufleuchten und kurze Zeit später eine der anderen roten LEDs (SL6) anfangen zu blinken, während eine der grünen LEDs (SL1) dauerhaft leuchtet. Dies bedeutet übrigens, dass sich kein Anwenderprogramm im Speicher des Controllers befindet. Wenn sich ein Programm im Speicher befindet, blinkt nur die grüne Status LED SL1.

Die gelbe PWRON LED sollte nur kurz für etwa eine Sekunde nach dem Anschalten leuchten und dann wieder ausgehen - es spart Energie die meisten nicht benötigten Sensoren wie z.B. die Drehgeber abzuschalten.

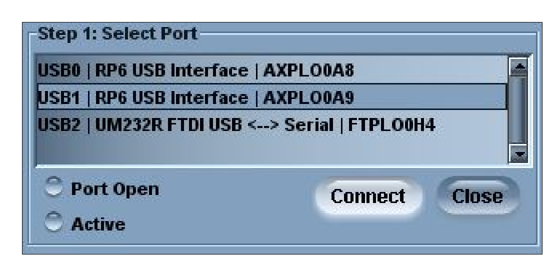
Nach etwa einer halben Minute sollte das Blinken der roten LED SL6 aufhören und alle LEDs ausgehen. Der Mikrocontroller auf dem Roboter schaltet sich nun - da er ohnehin noch kein Anwenderprogramm geladen hat das er ausführen könnte - automatisch in

den Standby aus dem er über das USB Interface oder einen Druck auf den Start/Stop Taster bzw. einmal aus und wieder anschalten wieder aufgeweckt werden kann. Der Roboter verbraucht auch in diesem Modus etwas Energie (bis zu 5mA) – also trotzdem immer daran denken den RP6 komplett auszuschalten, wenn er nicht verwendet wird! Wenn ein Programm geladen ist, schaltet sich der Roboter übrigens nicht von selbst in den Schlafmodus, sondern wartet immer auf Benutzereingaben bzw. den Startbefehl von der seriellen Schnittstelle (einfach ein „s" senden) oder über den I²C Bus.

3.8.1. USB Interface anschließen und RP6Loader starten

Als nächstes testen wir den Programmupload über das USB Interface. Verbinden Sie bitte das USB Interface mit dem PC **(Immer zuerst mit dem PC verbinden!)** und *danach* über das 10pol. Flachbandkabel mit dem seitlich neben dem Start/Stop Taster angeordneten "PROG/UART" Anschluss des RP6! Das 10 pol. Flachbandkabel ist mechanisch gegen Verpolung geschützt, sofern man es also nicht mit Gewalt behandelt, kann man es gar nicht verkehrt herum anschließen.

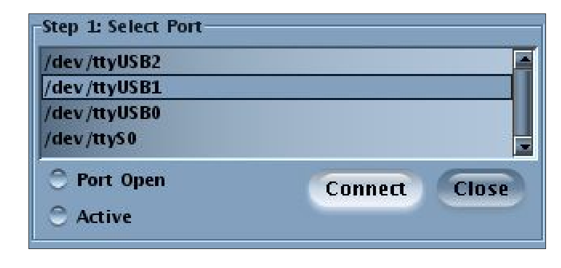
Starten Sie danach den RP6Loader. Je nachdem welche Sprache Sie gewählt haben, können die Menüs natürlich etwas anders beschriftet sein. Auf den Screenshots ist die englische Sprachversion dargestellt, über den Menüpunkt „Options->Preferences" und dann bei „Language / Sprache" kann man die Sprache anpassen (Englisch oder Deutsch) und danach auf OK klicken. Nach Änderung der Sprache muss man den RP6Loader aber erst einmal neu starten bevor sich etwas ändert!



Port öffnen - Windows

Jetzt können Sie den USB Port auswählen. Sofern kein anderer USB->Seriell Adapter mit FTDI Controller am PC angeschlossen ist, sehen sie in der Portliste nur einen einzigen Eintrag den Sie dann bitte auswählen. Falls doch mehrere Ports vorhanden sind, können Sie den Port anhand des Namens „RP6 USB Interface" identifizieren (oder „FT232R USB UART"). Dahinter wird noch die einprogrammierte Seriennummer angezeigt.

Sollten keine Ports angezeigt werden, können sie im Menü über „RP6Loader-->Refresh Portlist" („RP6Loader-->Portliste aktualisieren") die Portliste aktualisieren!



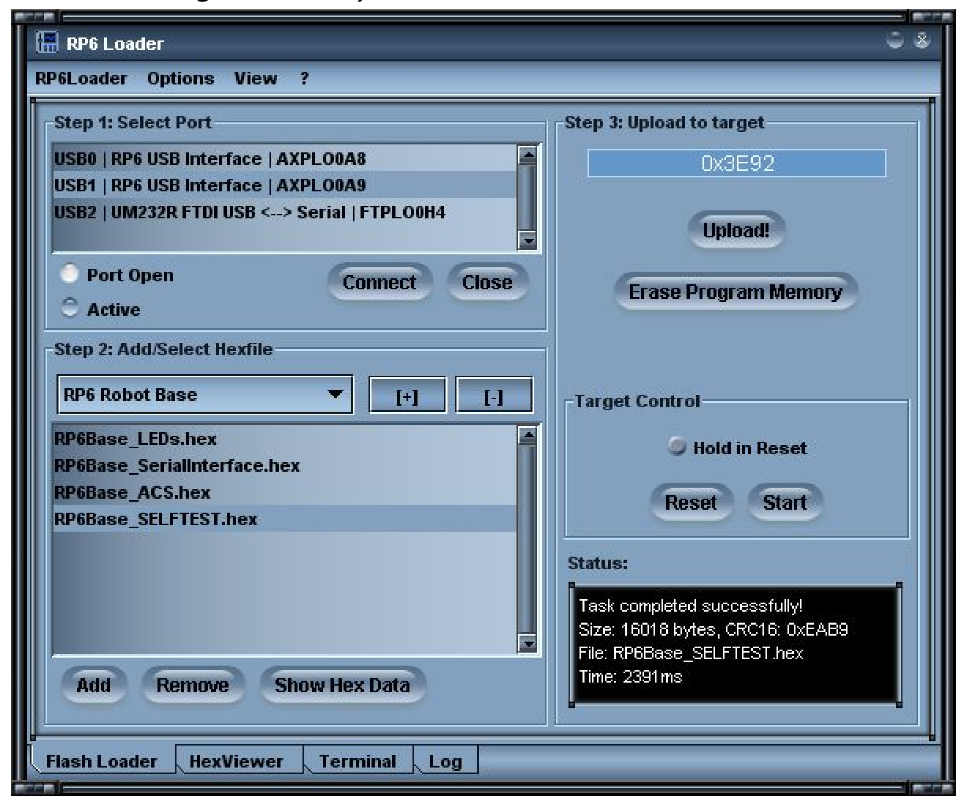
Port öffnen – Linux

Unter Linux wird der USB-Seriell Adapter wie ein normaler Comport behandelt. Die D2XX Treiberinstallation von FTDI unter Linux wäre nicht ganz so einfach und die normalen Virtual Comport (VCP) Treiber sind in aktuellen Linux Kerneln sowieso schon enthalten. Es funktioniert alles fast genauso wie unter Windows, nur muss man erst noch kurz ausprobieren welchen Namen das RP6 USB Interface hat und darauf achten, den USB Port nicht vom PC zu trennen solange die Verbindung noch offen ist (ansonsten muss der RP6Loader eventuell neu gestartet werden damit die Verbindung wieder klappt). Die Virtual Comports heissen un-

ter Linux „/dev/ttyUSBx", wobei x eine Nummer ist, z.B. „/dev/ttyUSB0" oder „/dev/ttyUSB1". Die normalen Comports heissen unter Linux „/dev/ttyS0", „/dev/ttyS1" etc.. Diese tauchen ebenfalls in der Portliste auf sofern vorhanden.

Der RP6Loader merkt sich – wenn es denn überhaupt mehrere Ports gibt - welchen Port Sie zuletzt verwendet haben und selektiert diesen bei jedem Start des Programms automatisch (generell bleiben die meisten Einstellungen und Selektionen erhalten).

Nun können Sie auf den Button „Connect" (=„Verbinden") klicken! Der RP6Loader öffnet nun den Port und testet, ob die Kommunikation mit dem Bootloader auf dem Roboter funktioniert. Es sollte dann unten im schwarzen Feld „Status" die Nachricht "Connected to: RP6 Robot Base ..." o.ä. zusammen mit einer Info über die aktuell gemessene Akkuspannung erscheinen. Falls nicht, probieren Sie es nochmal! Wenn es dann immer noch nicht klappt, liegt ein Fehler vor! Schalten Sie in diesem Fall den Roboter am Besten sofort aus und lesen Sie in Ruhe das Kapitel über Fehlersuche/Problemlösungen! Ist die Akkuspannung zu niedrig, erscheint eine Warnung. **Spätestens jetzt sollten die Akkus neu geladen werden** (besser schon dann, wenn die Spannung unter etwa 5.9V gefallen ist)!



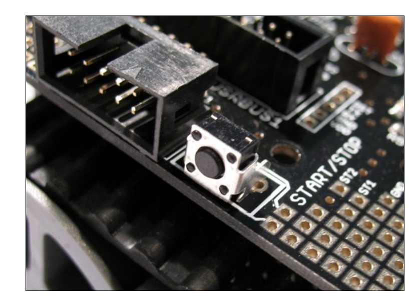
Wenn das geklappt hat, kann ein kleines Selbsttestprogramm ausgeführt werden um die Funktionsfähigkeit aller Systeme des Roboters zu überprüfen. Klicken Sie dazu unten im RP6Loader Fenster auf den Button „Add" („Hinzufügen") und wählen Sie die Datei „RP6Base_SELFTEST\RP6Base_SELFTEST.hex" im Beispielverzeichnis aus. In dieser Datei befindet sich das Selbsttestprogramm im hexadezimalen Format – daher werden solche Programmdateien auch „Hexdateien" genannt.

Die eben ausgewählte Datei taucht anschließend in der Liste auf. So können Sie später auch noch andere Hexdateien von Ihren eigenen Programmen und den Beispielprogrammen hinzufügen (s. Screenshot, hier wurden schon ein paar Hexdateien hin-

zugefügt). Der RP6Loader kann auch verschiedene Kategorien von Hexdateien verwalten. Damit lassen sich die Dateien übersichtlicher sortieren. Beispielsweise wenn man mehrere programmierbare Erweiterungsmodule auf dem RP6 hat, oder verschiedene Varianten von Programmen verwendet. Die Liste wird immer automatisch beim Beenden des Programms gespeichert! Es werden hier natürlich nur die Pfade zu den Hexdateien gespeichert – nicht die Hexdateien selbst. Wenn Sie an einem Programm arbeiten, brauchen Sie die Hexdatei nur einmal hinzufügen und auswählen und können danach sofort nach jedem erneuten Übersetzen des Programms, das neue Programm in den Mikrocontroller laden (auch per Tastenkombination [STRG+D] oder [STRG+Y] um das Programm direkt nach dem Übertragen zu starten). Unter verschiedenen Betriebssystemem sind die Pfadnamen natürlich komplett anders. Sie können den RP6Loader trotzdem ohne weitere Änderungen direkt unter Windows und Linux verwenden, denn es gibt für Windows und Linux jeweils eine extra Liste.

Wählen Sie jetzt die Datei „RP6Base_SELFTEST.hex" in der Liste aus und klicken Sie dann auf den Button „Upload!" oben rechts unter dem Fortschrittsbalken. Das Programm wird nun in den MEGA32 auf dem RP6 hochgeladen. Das sollte nicht länger als ein paar Sekunden dauern (maximal 5 Sekunden beim Selbsttest Programm).

Wechseln Sie anschließend bitte auf den Karteireiter (=„Tabs" ganz unten im Programmfenster!) „Terminal"! Alternativ geht das auch über das Menü „View" („Ansicht").



Starten Sie dann das Programm mit einem Druck auf den Start/Stop Taster auf dem RP6, neben dem Programmieranschluss (s. Abb.)! Später können Sie das natürlich auch alternativ über das Menü RP6Loader-->Start Target oder die Tastenkombination [STRG]+[S] tun, so können Sie allerdings direkt ausprobieren ob der Taster korrekt funktioniert! Es sollte zunächst eine Warnmeldung im Terminal erscheinen, in der darauf hingewiesen wird, dass der RP6 während des Tests Nummer 8 die Motoren anschaltet!

ACHTUNG! Halten Sie den RP6 bitte während des Tests Nummer 8 („Motors and Encoders Test") in den Händen oder stellen Sie einen passenden Gegenstand unter den RP6 – so dass die beiden Antriebsketten NICHT den Untergrund berühren können! ***Die Ketten dürfen während Test 8 NICHT berührt oder blockiert werden!*** Sonst schlägt der Test höchstwahrscheinlich fehl! Wenn der RP6 auf dem Boden stehen würde, würde sich das Verhalten des Antriebs ändern und der Test könnte dann nicht mehr korrekt funktionieren. Außerdem würde der RP6 ein gutes Stück weit fahren und man müsste das USB Kabel hinterhertragen – sofern es übehaupt lang genug dafür ist...

Also: den RP6 in die Hand nehmen oder etwas in der Mitte unter den RP6 stellen (z.B. eine kleine Schachtel oder eine Fernbedienung). Wenn Sie etwas unter den RP6 stellen, sollten Sie den RP6 aber trotzdem während des Tests mit einer Hand festhalten, damit er nicht wegrutscht und dann versehentlich vom Tisch fährt!

Die Warnmeldung erscheint auch direkt vor dem Test Nummer 8 nochmals und muss erst quittiert werden bevor es weitergeht. Genau wie jetzt, zu Beginn des Testprogramms. Geben Sie bitte ein kleines 'x' im Terminal ein und drücken Sie die Enter Taste (Das müssen Sie noch häufiger tun! Immer wenn so eine Meldung erscheint oder um einen Test abzubrechen...).

```
######################################################################
#########            RP6 Robot Base Selftest             #########
######### HOME VERSION                 v. 1.0 - 12.02.2007 #########
######################################################################
#####        Main Menu         #########      Advanced Menu      #####
#                                  #                                 #
# 0 - Run ALL Selftests (0-8)      # s - Move at speed Test          #
# 1 - PowerOn Test                 # d - Move distance Test          #
# 2 - LED Test                     # c - Encoder Duty-Cycle Test     #
# 3 - Voltage Sensor Test          #                                 #
# 4 - Bumper Test                  #                                 #
# 5 - Light Sensor Test            #                                 #
# 6 - ACS (and RC5 receive) Test   #                                 #
# 7 - IRCOMM/RC5 Test              #                                 #
# 8 - Motors and Encoders Test     #                                 #
#                                  #                                 #
######################################################################
# Please enter your choice (1-8, s, d, c)!                           #
######################################################################
```

Jetzt sollte in etwa das Textmenü auf der linken Seite erscheinen!

Das kann sich aber noch bis zur finalen Software Version etwas verändern.

Sie können die Testprogramme durch Eingabe der entsprechenden Zahl bzw. des Buchstabens auswählen und starten.

Wir wollen alle Standard Tests durchführen – also geben Sie bitte eine '0' ein und drücken Enter!

Es sollten folgenden Ausgaben im Teminal erscheinen:

```
# 0

######################################################################
######################################################################
## Test #1 ##

### POWER ON TEST ###
Please watch the yellow PowerOn LED and verify that it lights up!
(it will flash a few times!)
```

Schauen Sie sich jetzt die gelbe PowerON LED auf dem RP6 an! Diese sollte nun ein paar mal an und aus gehen. Tut Sie das nicht, ist entweder der Test schon vorbei bevor Sie auf die LED geschaut haben, oder es ist tatsächlich etwas defekt. Das Testprogramm fährt aber trotzdem weiter fort – denn es kann nicht selbst feststellen ob die LED funktioniert oder nicht – das müssen Sie selbst überprüfen!

Die LED zeigt übrigens an, dass die Drehgeber, der IR Empfänger und die Stromsensoren eingeschaltet sind. Zusammen mit der LED benötigen diese insgesamt immerhin knapp 10mA --> daher wird das alles zum Stromsparen nur bei Bedarf angeschaltet.

Das Programm wird nun alle Status LEDs aufleuchten lassen. Ein paar mal alle zusammen und ein paar mal einzeln. Hier können Sie sofort sehen ob alle LEDs korrekt funktionieren oder evtl. beschädigt sind. Ausgabe:

```
## Test #2 ##

### LED Test ###
Please watch the LEDs and verify that they all work!
Done!
```

Jetzt folgt der Akkusensortest. Der Sensor wurde eigentlich schon getestet, weil der RP6Loader die Akkuspannung ausliest und anzeigt, aber hier nochmal der Vollständigkeit halber.

```
######################################################################
######################################################################
## Test #3 ##

### Voltage Sensor Test ###
Be sure that you are using good accumulators!

Enter "x" and hit return when you are ready!
```

Bestätigen Sie bitte mit 'x'!

```
# x
Performing 10 measurements:
Measurement #1: 07.20V --> OK!
Measurement #2: 07.20V --> OK!
Measurement #3: 07.20V --> OK!
Measurement #4: 07.20V --> OK!
Measurement #5: 07.20V --> OK!
Measurement #6: 07.20V --> OK!
Measurement #7: 07.20V --> OK!
Measurement #8: 07.20V --> OK!
Measurement #9: 07.20V --> OK!
Measurement #10: 07.20V --> OK!
Done!
```

So sollte die Ausgabe in etwa aussehen – die Spannungswerte dürfen im Bereich von 5.5 bis 9.5V liegen – dann wird der Messwert als OK angesehen. Liegt der Messwert ausserhalb dieses Bereichs, wird eine Fehlermeldung angezeigt. Bitte überprüfen Sie in diesem Fall die Akkus – evtl. sind sie nicht aufgeladen oder defekt! Wenn die Akkus OK sind, kann aber auch der „Sensor“ (zwei Widerstände...) beschädigt sein.

Jetzt testen wir die Bumpersensoren. Dazu müssen Sie einfach ein wenig auf den Bumpern herumdrücken und sich die LEDs bzw. die Ausgaben auf dem Terminal ansehen. Bei jedem Druck auf einen der Bumper und wenn dieser wieder losgelassen wird, sollte eine Ausgabe im Terminal erscheinen und sich der Zustand der LEDs ändern! Das könnte in etwa so ausschauen:

```
## Test #4 ##

Bumper Test
Please hit both bumpers and verify
that both Bumpers are working properly!
The Test is running now. Enter "x" and hit return to stop this test!
OBSTACLE: LEFT!
FREE: LEFT!
OBSTACLE: RIGHT!
FREE: RIGHT!
OBSTACLE: LEFT!
FREE: LEFT!
OBSTACLE: RIGHT!
FREE: RIGHT!
OBSTACLE: LEFT!
OBSTACLE: RIGHT!
FREE: LEFT!
FREE: RIGHT!
```

Den Test können Sie mit 'x' + Enter abbrechen wenn alles geklappt hat!

Es folgt der Lichtsensor Test. Hier müssen Sie die beiden Lichtsensoren vorne am Roboter mit den Händen abdunkeln und darauf achten, dass sich die Messwerte entsprechend verändern – je kleiner die Messwerte, desto weniger Licht fällt auf die Sensoren! Bei normalem Tageslicht sollten die Werte etwa zwischen 200 und 900 liegen.

Wenn man mit einer starken Taschenlampe ganz nah herangeht und direkt auf die Sensoren leuchtet oder den Roboter in die Sonne hält, können die Werte bis etwa 1000 ansteigen. Ist es im Raum sehr dunkel, sollten die Werte unter 100 liegen.

Zu Beginn des Tests muss man wieder mit x bestätigen:

```
## Test #5 ##

### Light Sensor Test ###
Please get yourself a small flashlight!
While the test runs, move it in front of the Robot
and watch if the values change accordingly!

Enter "x" and hit return when you are ready!
# x
The Test is running now. Enter "x" and hit return to stop this test!
Performing measurements...:
Left: 0510,  Right: 0680
Left: 0511,  Right: 0679
Left: 0512,  Right: 0680
Left: 0560,  Right: 0710
Left: 0630,  Right: 0750
Left: 0640,  Right: 0760
Left: 0644,  Right: 0765

[...]
```

Den Test dann bitte mit Eingabe von 'x' abbrechen, wenn Sie die Sensoren getestet haben!

Dann geht es direkt weiter zum ACS Test. Hier muss nichts bestätigt werden, der Test fängt sofort an. Bewegen Sie also nun ein wenig eine Hand oder Gegenstände vor dem Roboter hin und her. Dabei sollte vor dem Roboter allerdings viel Platz sein damit die Sensoren nicht dauerhaft ein Hindernis detektieren.

Die Ausgabe sollte in etwa so aussehen:

```
## Test #6 ##

ACS Test
Please move your hand or other obstacles in front of the Robot
and verify that both ACS channels are working properly!

ACS is set to Medium power/range!

You can also send RC5 Codes with a TV Remote Control
to the RP6 - it will display the Toggle Bit, Device Adress
and Keycode of the RC5 Transmission!
Make sure your remote control transmits in RC5 and not
SIRCS or RECS80 etc.! There are several other formats that will NOT work!

The Test is running now. Enter "x" and hit return to stop this test!
OBSTACLE: LEFT!
FREE: LEFT!
OBSTACLE: LEFT!
FREE: LEFT!
OBSTACLE: LEFT!
OBSTACLE: RIGHT!
FREE: RIGHT!
FREE: LEFT!
OBSTACLE: LEFT!
OBSTACLE: RIGHT!
FREE: RIGHT!
FREE: LEFT!
```

Sie können während des Tests auch Codes einer RC5 kompatiblen IR Fernbedienung empfangen. Es werden dann Togglebit, Adresse und Keycode angezeigt.

Um den Test abzubrechen einfach x eingeben und Enter drücken.

Weiter geht's mit dem IRCOMM Test. Hier muss zunächst wieder mit x bestätigt werden. Dann werden IR Datenpakete gesendet und alle empfangenen Daten im Terminal ausgegeben und automatisch überprüft (das IRCOMM empfängt normalerweise auch die von ihm selbst gesendeten Signale, da die IR Dioden recht stark sind... wenn natürlich überhaupt keine reflektierenden Gegenstände bzw. die Decke über dem Roboter ist, könnte das auch mal nicht korrekt funktionieren – ist aber unwahrscheinlich).

Die Ausgaben des Tests sollten etwa wie folgt aussehen:

```
#### TEST #7 ####

IRCOMM Test
[...]

TX RC5 Packet: 0
RX RC5 Packet --> Toggle Bit:0 | Device Address:0 | Key Code:0 --> OK!
TX RC5 Packet: 3
RX RC5 Packet --> Toggle Bit:0 | Device Address:3 | Key Code:3 --> OK!
TX RC5 Packet: 6
RX RC5 Packet --> Toggle Bit:0 | Device Address:6 | Key Code:6 --> OK!
TX RC5 Packet: 9
RX RC5 Packet --> Toggle Bit:0 | Device Address:9 | Key Code:9 --> OK!
TX RC5 Packet: 12
RX RC5 Packet --> Toggle Bit:0 | Device Address:12 | Key Code:12 --> OK!
[...]
TX RC5 Packet: 57
RX RC5 Packet --> Toggle Bit:1 | Device Address:25 | Key Code:57 --> OK!
TX RC5 Packet: 60
RX RC5 Packet --> Toggle Bit:1 | Device Address:28 | Key Code:60 --> OK!
TX RC5 Packet: 63
RX RC5 Packet --> Toggle Bit:1 | Device Address:31 | Key Code:63 --> OK!

Test finished!
Done!
```

Der Test sollte nur etwa 5 Sekunden dauern.

Jetzt kommt der Motor und Encoder Test! Sie müssen den RP6 also unbedingt in die Hand nehmen – die Ketten dürfen den Boden oder andere Objekte nicht berühren!

Sonst wird der Test vermutlich fehlschlagen! Passen Sie auf, dass der RP6 nicht vom Tisch fällt, wenn Sie wie oben beschrieben einen Gegenstand unter den RP6 stellen wollen.

Der Test dauert nicht besonders lange – etwa 30 Sekunden. Achten Sie unbedingt auf eventuelle Fehlermeldungen während des Tests! Es kann schonmal passieren, dass einzelne Messwerte fehlerhaft sind und der Test deswegen fehlschlägt. Also wenn die Motoren normal anlaufen und der Test erst irgendwann mittendrin abbricht, muss das noch nichts schlimmes bedeuten. In diesem Fall einfach diesen Test nochmal durchführen – aber vorher auch das Kapitel über Fehlerbehandlung im Anhang lesen!

Die Testroutine lässt den Antrieb nacheinander in verschiedenen Geschwindigkeitsstufen bis etwa 50% der Maximalgeschwindigkeit laufen und wechselt später auch ein paar mal die Laufrichtung der Motoren. Dabei werden ständig die Messwerte von den Drehgebern und den Stromsensoren überwacht. Sollte z.B. beim Transport etwas beschädigt worden sein (z.B. ein Kurzschluss in einem der Motoren oder ein verklemm-

tes Getriebe – das sollte man aber eigentlich schon oben beim Test vor dem Akkueinbau bemerkt haben) wird der gemessene Strom sehr hoch sein und der Test sofort abbrechen.

So sollte das dann in etwa aussehen (etwas gekürzt):

```
#####################################################################
#####################################################################
#### TEST #8 ####

Automatic speed speed regulation test

#####################################################################
### ATTENTION!!! DANGER!!! WARNING!!!
Make sure that the RP6 can __NOT__ move!
The caterpillar tracks should __NOT__ touch the ground!
(hold it in your hands for example...)
THE RP6 WILL START MOVING FAST! YOU CAN DAMAGE IT IF YOU DO NOT
MAKE SURE THAT IT CAN __NOT__ MOVE!
Make sure both crawler tracks are FREE RUNNING! DO NOT BLOCK THEM!
--> OTHERWISE THE TEST WILL FAIL!
#####################################################################

Enter "x" and hit return when TO START THIS TEST!
Make sure the RP6 can not move!

# x
T: 000 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 000 |IR: 003 |UB: 07.28V
T: 000 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 002 |IR: 003 |UB: 07.28V
[...]
Speed Left: OK
Speed Right: OK
T: 020 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 000 |IR: 003 |UB: 07.28V
T: 020 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 000 |IR: 003 |UB: 07.28V
T: 020 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 000 |IR: 003 |UB: 07.28V
T: 020 |VL: 000 |VR: 000 |PL: 020 |PR: 020 |IL: 006 |IR: 009 |UB: 07.26V
T: 020 |VL: 001 |VR: 014 |PL: 039 |PR: 030 |IL: 020 |IR: 020 |UB: 07.27V
[...]
Speed Left: OK
Speed Right: OK
T: 040 |VL: 021 |VR: 019 |PL: 037 |PR: 028 |IL: 025 |IR: 021 |UB: 07.25V
T: 040 |VL: 020 |VR: 020 |PL: 037 |PR: 029 |IL: 026 |IR: 022 |UB: 07.25V
T: 040 |VL: 018 |VR: 020 |PL: 044 |PR: 036 |IL: 028 |IR: 023 |UB: 07.23V
T: 040 |VL: 038 |VR: 038 |PL: 055 |PR: 044 |IL: 035 |IR: 029 |UB: 07.23V
T: 040 |VL: 037 |VR: 042 |PL: 055 |PR: 043 |IL: 033 |IR: 028 |UB: 07.24V
T: 040 |VL: 043 |VR: 041 |PL: 052 |PR: 042 |IL: 032 |IR: 026 |UB: 07.23V
T: 040 |VL: 043 |VR: 041 |PL: 052 |PR: 040 |IL: 030 |IR: 024 |UB: 07.24V
T: 040 |VL: 037 |VR: 041 |PL: 052 |PR: 040 |IL: 030 |IR: 023 |UB: 07.24V
T: 040 |VL: 043 |VR: 040 |PL: 050 |PR: 039 |IL: 029 |IR: 022 |UB: 07.24V
Speed Left: OK
Speed Right: OK
T: 060 |VL: 040 |VR: 039 |PL: 053 |PR: 040 |IL: 033 |IR: 024 |UB: 07.24V
T: 060 |VL: 036 |VR: 040 |PL: 053 |PR: 040 |IL: 034 |IR: 026 |UB: 07.24V
T: 060 |VL: 042 |VR: 039 |PL: 052 |PR: 041 |IL: 034 |IR: 027 |UB: 07.23V
T: 060 |VL: 042 |VR: 040 |PL: 063 |PR: 052 |IL: 038 |IR: 032 |UB: 07.22V
T: 060 |VL: 058 |VR: 060 |PL: 068 |PR: 056 |IL: 038 |IR: 032 |UB: 07.25V
T: 060 |VL: 062 |VR: 062 |PL: 067 |PR: 054 |IL: 037 |IR: 029 |UB: 07.22V
T: 060 |VL: 060 |VR: 062 |PL: 067 |PR: 053 |IL: 038 |IR: 028 |UB: 07.23V

[...]

Speed Left: OK
Speed Right: OK
T: 100 |VL: 082 |VR: 078 |PL: 080 |PR: 068 |IL: 043 |IR: 036 |UB: 07.23V
T: 100 |VL: 079 |VR: 079 |PL: 081 |PR: 069 |IL: 047 |IR: 038 |UB: 07.22V
T: 100 |VL: 078 |VR: 082 |PL: 092 |PR: 078 |IL: 049 |IR: 039 |UB: 07.23V
T: 100 |VL: 095 |VR: 099 |PL: 101 |PR: 082 |IL: 055 |IR: 039 |UB: 07.20V
T: 100 |VL: 098 |VR: 100 |PL: 109 |PR: 081 |IL: 056 |IR: 040 |UB: 07.19V
T: 100 |VL: 095 |VR: 099 |PL: 111 |PR: 082 |IL: 062 |IR: 042 |UB: 07.19V
T: 100 |VL: 102 |VR: 101 |PL: 111 |PR: 082 |IL: 058 |IR: 041 |UB: 07.21V
T: 100 |VL: 102 |VR: 101 |PL: 109 |PR: 081 |IL: 056 |IR: 039 |UB: 07.20V
T: 100 |VL: 093 |VR: 100 |PL: 113 |PR: 081 |IL: 063 |IR: 038 |UB: 07.20V
```

```
T: 100 |VL: 104 |VR: 099 |PL: 112 |PR: 082 |IL: 056 |IR: 042 |UB: 07.22V
Speed Left: OK
Speed Right: OK
T: 080 |VL: 086 |VR: 071 |PL: 022 |PR: 000 |IL: 020 |IR: 012 |UB: 07.28V
T: 080 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 001 |IR: 003 |UB: 07.28V
T: 080 |VL: 004 |VR: 011 |PL: 088 |PR: 084 |IL: 051 |IR: 045 |UB: 07.21V
T: 080 |VL: 079 |VR: 101 |PL: 103 |PR: 077 |IL: 064 |IR: 039 |UB: 07.21V
T: 080 |VL: 082 |VR: 076 |PL: 098 |PR: 072 |IL: 061 |IR: 041 |UB: 07.19V
T: 080 |VL: 081 |VR: 081 |PL: 096 |PR: 071 |IL: 055 |IR: 040 |UB: 07.20V
T: 080 |VL: 080 |VR: 082 |PL: 095 |PR: 070 |IL: 057 |IR: 038 |UB: 07.21V
T: 080 |VL: 082 |VR: 080 |PL: 094 |PR: 069 |IL: 058 |IR: 036 |UB: 07.22V
T: 080 |VL: 077 |VR: 080 |PL: 095 |PR: 069 |IL: 056 |IR: 036 |UB: 07.23V
Speed Left: OK
Speed Right: OK
T: 060 |VL: 082 |VR: 079 |PL: 095 |PR: 069 |IL: 054 |IR: 038 |UB: 07.22V
T: 060 |VL: 079 |VR: 079 |PL: 095 |PR: 071 |IL: 058 |IR: 040 |UB: 07.21V
T: 060 |VL: 082 |VR: 081 |PL: 093 |PR: 070 |IL: 056 |IR: 039 |UB: 07.19V
T: 060 |VL: 069 |VR: 070 |PL: 080 |PR: 054 |IL: 048 |IR: 029 |UB: 07.23V
T: 060 |VL: 064 |VR: 059 |PL: 075 |PR: 054 |IL: 046 |IR: 029 |UB: 07.22V
T: 060 |VL: 058 |VR: 057 |PL: 075 |PR: 055 |IL: 043 |IR: 032 |UB: 07.24V
T: 060 |VL: 059 |VR: 059 |PL: 075 |PR: 056 |IL: 046 |IR: 034 |UB: 07.23V
T: 060 |VL: 060 |VR: 059 |PL: 075 |PR: 056 |IL: 046 |IR: 035 |UB: 07.23V
T: 060 |VL: 057 |VR: 060 |PL: 076 |PR: 056 |IL: 047 |IR: 033 |UB: 07.22V
T: 060 |VL: 058 |VR: 061 |PL: 077 |PR: 055 |IL: 045 |IR: 030 |UB: 07.23V
Speed Left: OK
Speed Right: OK
T: 040 |VL: 045 |VR: 035 |PL: 043 |PR: 023 |IL: 027 |IR: 018 |UB: 07.24V
T: 040 |VL: 000 |VR: 000 |PL: 011 |PR: 000 |IL: 013 |IR: 007 |UB: 07.28V
T: 040 |VL: 002 |VR: 000 |PL: 038 |PR: 038 |IL: 015 |IR: 014 |UB: 07.24V
T: 040 |VL: 038 |VR: 061 |PL: 059 |PR: 052 |IL: 035 |IR: 035 |UB: 07.24V
T: 040 |VL: 044 |VR: 043 |PL: 057 |PR: 044 |IL: 035 |IR: 028 |UB: 07.23V
T: 040 |VL: 038 |VR: 039 |PL: 057 |PR: 044 |IL: 035 |IR: 027 |UB: 07.24V
T: 040 |VL: 039 |VR: 042 |PL: 055 |PR: 043 |IL: 033 |IR: 025 |UB: 07.23V
T: 040 |VL: 043 |VR: 041 |PL: 053 |PR: 041 |IL: 032 |IR: 023 |UB: 07.24V
T: 040 |VL: 040 |VR: 041 |PL: 054 |PR: 041 |IL: 032 |IR: 023 |UB: 07.25V
Speed Left: OK
Speed Right: OK
T: 020 |VL: 037 |VR: 040 |PL: 054 |PR: 041 |IL: 031 |IR: 024 |UB: 07.24V
T: 020 |VL: 022 |VR: 019 |PL: 022 |PR: 012 |IL: 017 |IR: 016 |UB: 07.28V
T: 020 |VL: 000 |VR: 000 |PL: 000 |PR: 000 |IL: 004 |IR: 007 |UB: 07.28V
T: 020 |VL: 000 |VR: 006 |PL: 030 |PR: 027 |IL: 020 |IR: 020 |UB: 07.24V
T: 020 |VL: 013 |VR: 019 |PL: 043 |PR: 030 |IL: 029 |IR: 022 |UB: 07.24V
T: 020 |VL: 026 |VR: 020 |PL: 038 |PR: 029 |IL: 027 |IR: 022 |UB: 07.24V
T: 020 |VL: 020 |VR: 021 |PL: 038 |PR: 029 |IL: 028 |IR: 023 |UB: 07.25V
T: 020 |VL: 021 |VR: 020 |PL: 038 |PR: 029 |IL: 028 |IR: 023 |UB: 07.24V
T: 020 |VL: 018 |VR: 019 |PL: 038 |PR: 030 |IL: 027 |IR: 024 |UB: 07.24V
T: 020 |VL: 022 |VR: 020 |PL: 037 |PR: 029 |IL: 027 |IR: 023 |UB: 07.23V
Speed Left: OK
Speed Right: OK

***** MOTOR AND ENCODER TEST OK! *****
```

Die einzelnen Messwerte die während der Tests ausgegeben werden sind (von links nach rechts): T – aktueller Geschwindigkeitssollwert, VL/VR - gemessene Geschwindigkeit links/rechts, PL/PR – PWM Wert links/rechts, IL/IR – Motorstrom links/rechts, UB – Akkuspannung.

Wenn die Ausgaben ***ähnlich*** wie oben aussehen – ist alles OK.

Wenn etwas nicht korrekt funktionieren sollte und Fehlermeldungen erscheinen, lesen Sie bitte den Abschnitt zur Fehlerbehandlung im Anhang!

Das war es auch schon mit dem Testprogramm. Wenn alles wie erwartet geklappt hat, können Sie direkt mit dem nächsten Abschnitt weitermachen.

4. Programmierung des RP6

Nun kommen wir so langsam zur Programmierung des Roboters.

4.1. Einrichten des Quelltexteditors

Erstmal müssen wir uns eine kleine Entwicklungsumgebung einrichten. Der sog. „Quelltext" (auch Quellcode oder engl. Sourcecode genannt) für unsere C Programme muss ja irgendwie in den Computer eingegeben werden!

Dazu werden wir natürlich auf gar keinen Fall Programme wie OpenOffice oder Word verwenden! Vielleicht ist das nicht für jeden offensichtlich, deshalb wird es hier explizit betont. Damit kann man zwar gut Handbücher wie dieses hier schreiben, aber zum Programmieren ist das absolut ungeeignet. Quelltext ist reiner Text – ohne jegliche Formatierung. Schriftgröße oder Farbe interessieren den Compiler nicht...

Für einen Menschen ist es natürlich übersichtlicher wenn bestimmte Schlüsselwörter oder Arten von Text automatisch farbig hervorgehoben werden. Das und noch einiges mehr bietet Programmers Notepad 2 (im folgenden kurz „PN2" genannt), der Quelltexteditor den wir verwenden werden *(ACHTUNG: Unter Linux müssen Sie einen anderen Editor verwenden, der ähnliches wie PN2 bietet. Es sind für gewöhnlich mehrere bereits vorinstalliert! (z.B. kate, gedit, exmacs o.ä.))*. Neben dem Hervorheben von Schlüsselwörtern und ähnlichem (sog. „Syntax Highlighting") gibt es auch eine rudimentäre Projektverwaltung. Man kann so mehrere Quelltextdateien in Projekten organisieren und in einer Liste alle zu einem Projekt gehörenden Dateien anzeigen lassen. Weiterhin kann man aus PN2 komfortabel Programme wie den AVR-GCC aufrufen und so die Programme bequem über einen Menüeintrag übersetzen lassen. Der AVR-GCC ist ja normalerweise ein reines Kommandozeilenprogramm ohne graphische Oberfläche...

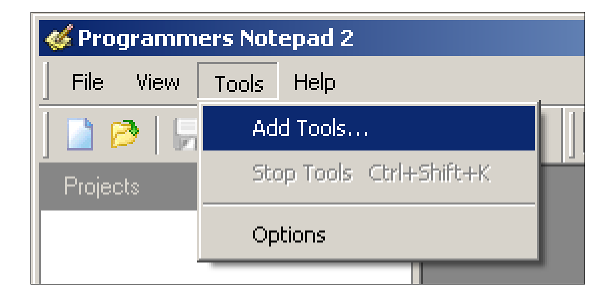
Neuere Versionen von Programmers Notepad finden Sie auf der Projekthomepage:

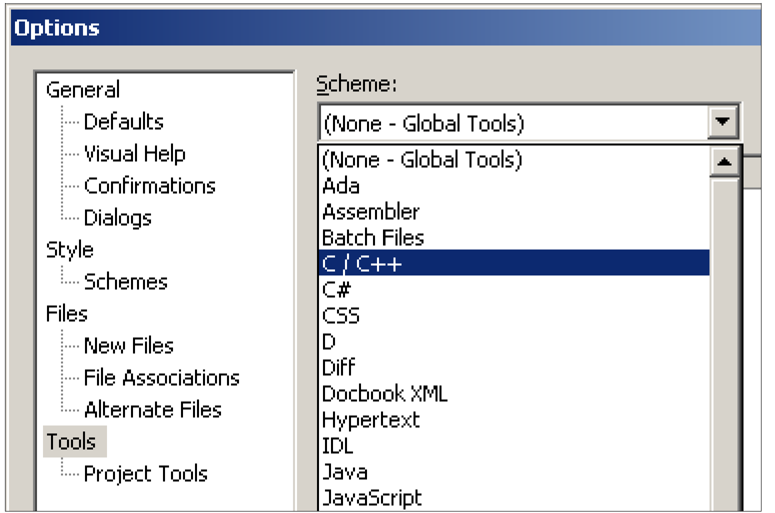
`http://www.pnotepad.org/`

4.1.1. Menüeinträge erstellen

ACHTUNG: Diesen Abschnitt können Sie überspringen wenn in PN2 die Menüeinträge schon vorhanden sind (die Menüeinträge heissen dann „[WinAVR] Make All", etc.. einfach mal in dem Menü nachschauen). Da dies aber nicht bei allen Versionen der Fall ist und es außerdem nützlich ist zu wissen wie man das macht (Sie können ja auch andere Programme wie den RP6Loader in dieses Menü einfügen!), beschreiben wir hier noch kurz wie man eigene Menüeinträge hinzufügen kann.

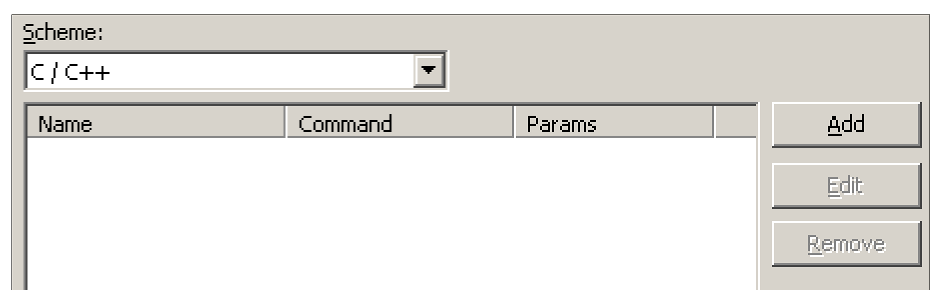
Starten Sie PN2 und wählen Sie dann im Menü „Tools" den Menüpunkt „Add Tools..." aus (s. Screenshot).



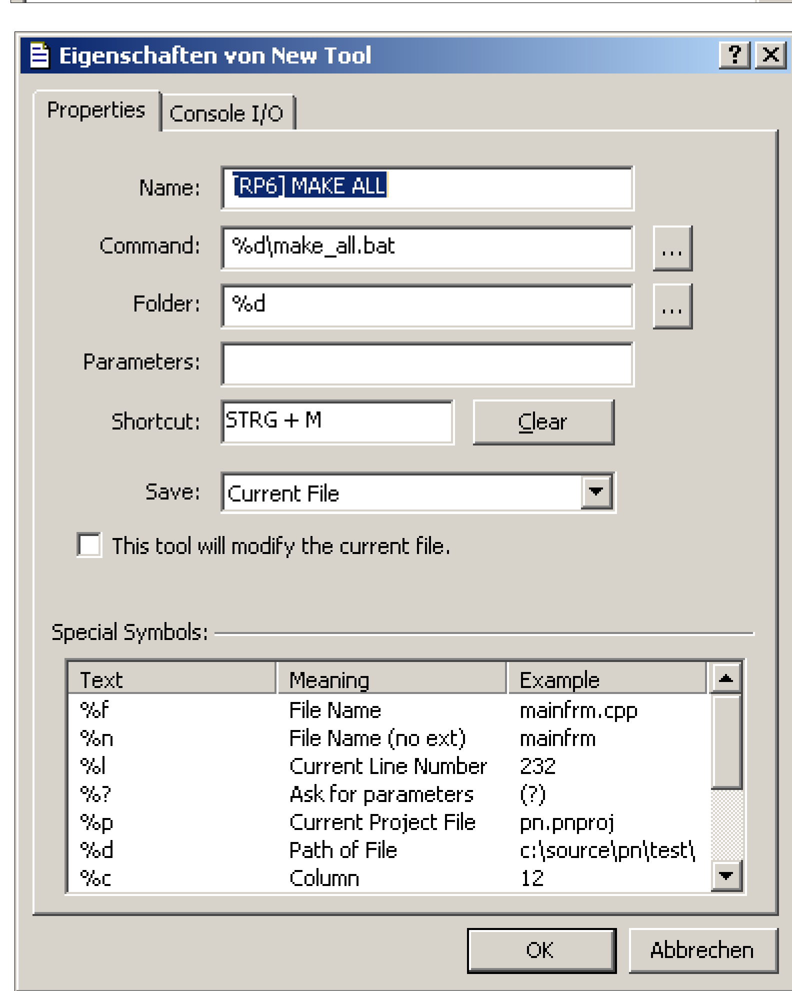


Es erscheint der Optionen Dialog. Hier können Sie diverse Einstellungen von PN2 verändern, wir wollen aber erstmal nur neue Menüeinträge zum Tools Menü hinzufügen.

Wählen Sie dazu in der „Scheme:" Dropdown Liste „C/C++" aus!



Klicken Sie dann auf den Button „Add"!



Es erscheint der nebenstehende Dialog.

Hier geben Sie bitte genau das ein, was Sie auf dem Screenshot sehen.

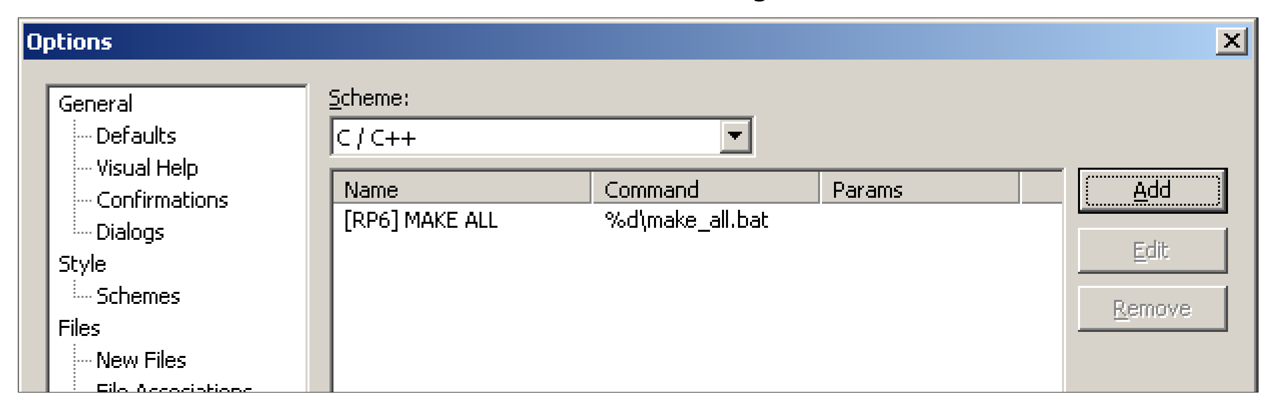
„%d" verweisst auf das Verzeichnis der aktuell gewählten Datei und „%d\make_all.bat" verweisst auf eine kleine Batchdatei, die in jedem Verzeichnis der vorgefertigten Projekte für den RP6 zu finden ist.

Bei „Shortcut" können Sie Beispielsweise [STRG] + [M] auf der Tastatur drücken! Das ist aber optional!

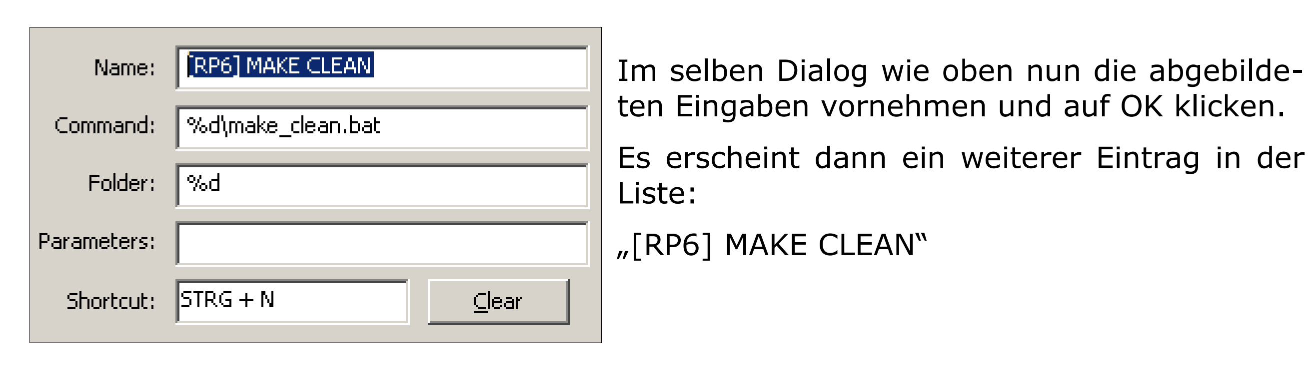
Dieser Eintrag ruft das Tool „make" über die „make_all.bat" Batchdatei auf und leitet so das Kompilieren der Dateien im Verzeichnis der aktuell gewählten Datei ein. Aber dazu später mehr.

Alternativ zu „%d/make_all.bat“ kann man auch einfach nur „make“ ins Feld „Command“ schreiben und bei Parameters „all“. Die Batch Datei führt eigentlich auch nur diese Befehle aus (ist aber nützlich um das Programm einfach so aus dem Windows Explorer heraus zu kompilieren).

Jetzt auf OK klicken – es erscheint ein neuer Eintrag in der Liste:



...und nochmal auf „Add“ klicken!



Im selben Dialog wie oben nun die abgebildeten Eingaben vornehmen und auf OK klicken.

Es erscheint dann ein weiterer Eintrag in der Liste:

„[RP6] MAKE CLEAN“

Mit diesem Eintrag kann man die ganzen temporären Dateien die der Compiler während seiner Arbeit erzeugt hat komfortabel löschen. Die brauchen wir nämlich nach dem Kompilieren meistens nicht mehr. Die erzeugte Hexdatei wird übrigens nicht gelöscht und kann noch immer in den Roboter geladen werden.

Wie oben kann man alternativ zu „%d/make_clean.bat“ auch einfach nur „make“ ins Feld „Command“ schreiben und bei Parameters „clean“.

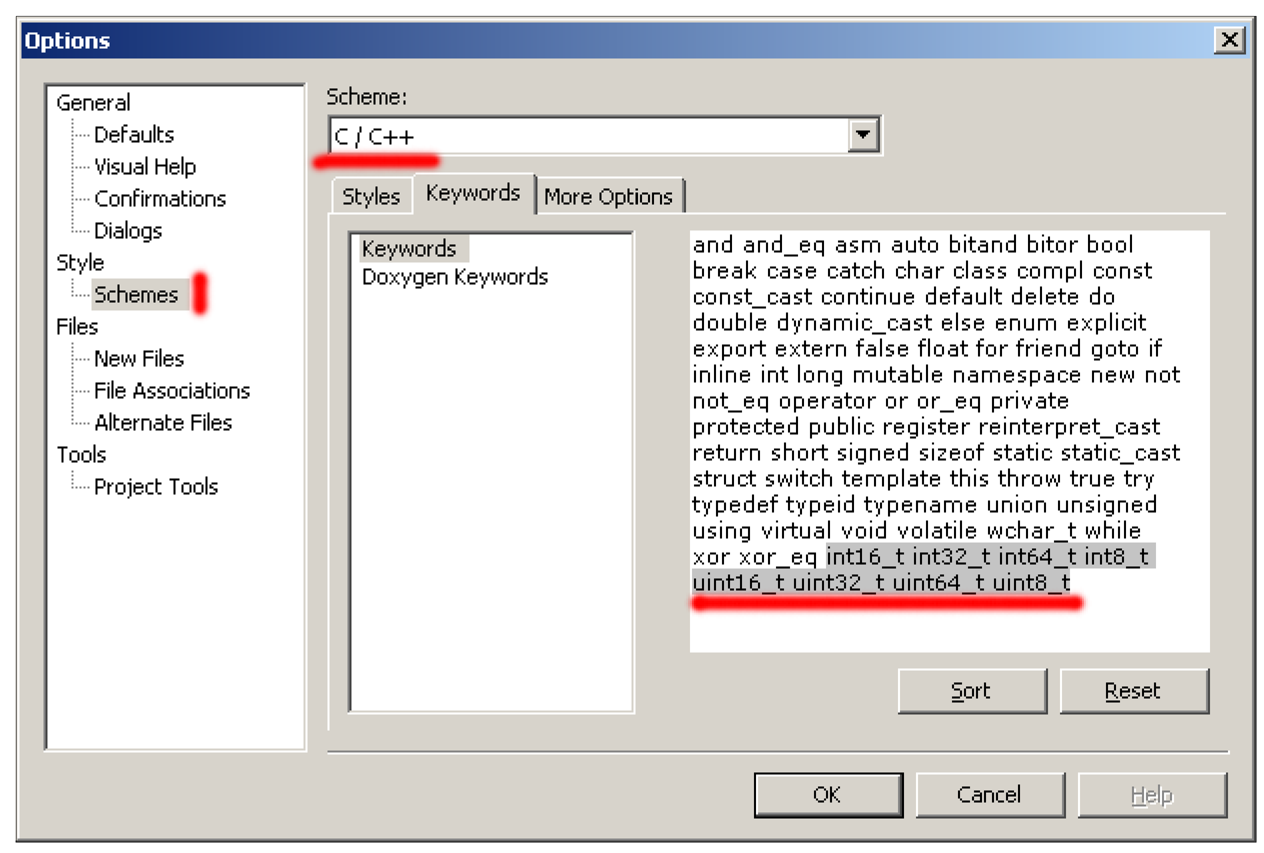
Abschließend müssen Sie noch die Änderungen im „Options“ Dialog mit einem Klick auf „OK“ bestätigen!

4.1.2. Syntax Highlighting einstellen

Eine weitere Einstellung die Sie verändern sollten betrifft das Syntax Highlighting. Es gibt einige „Keywords“ (=Schlüsselwörter) die dem normalen C/C++ Schema noch hinzugefügt werden sollten.

Das sind folgende Keywords (können Sie hier direkt per Copy & Paste ([STRG]+[C] = kopieren, [STRG]+[V] = einfügen) in das Dialogfeld kopieren!):

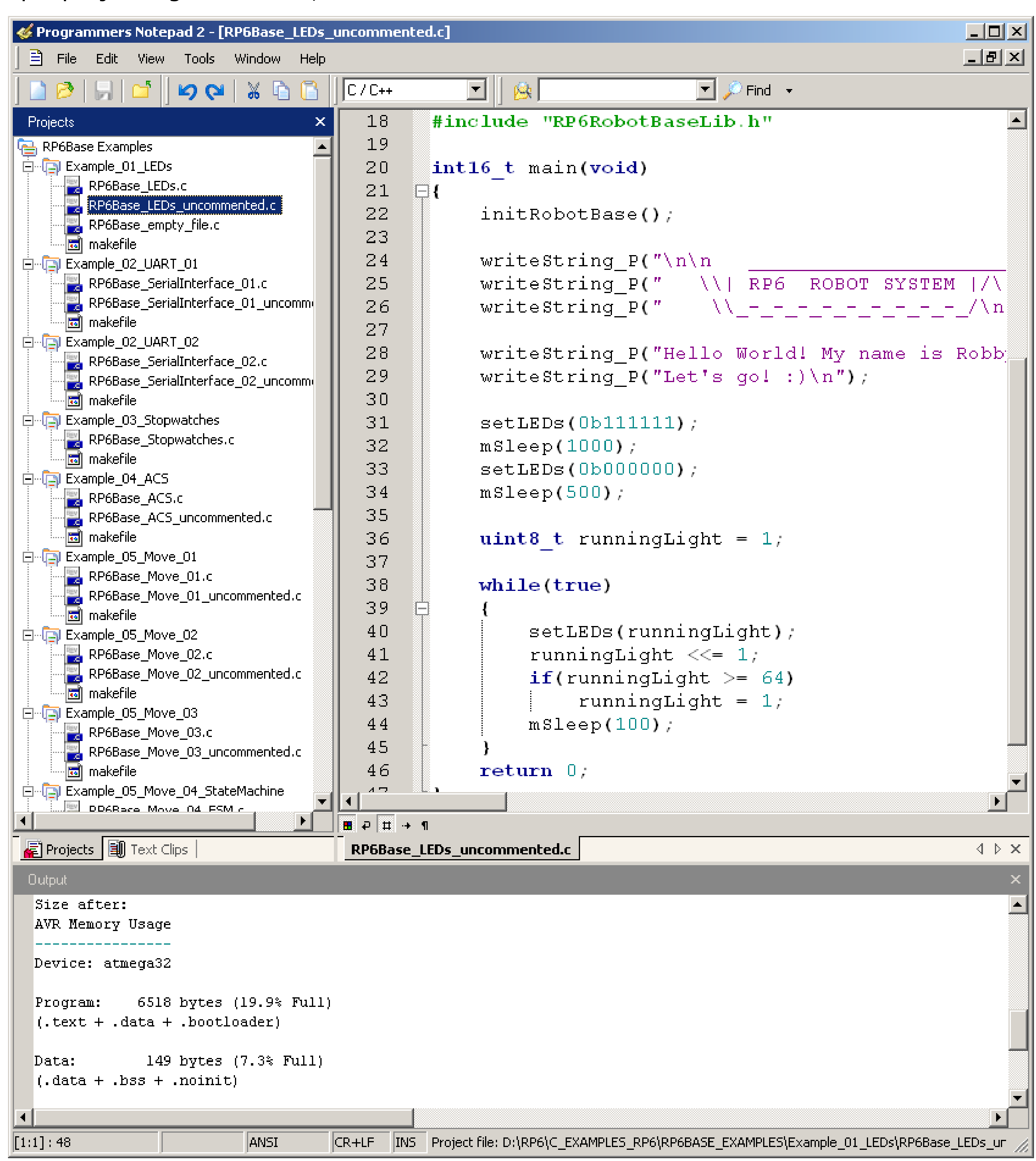
int8_t int16_t int32_t int64_t uint8_t uint16_t uint32_t uint64_t



Dann einmal auf „Sort“ (Sortieren) klicken und den Dialog mit OK bestätigen!

Achtung: *Bei neueren WinAVR und Programmers Notepad Versionen (WinAVR-20070525) sind diese Keywords bereits in Programmers Notepad eingetragen! Dann brauchen Sie hier nichts mehr zu ändern! In dieser Version sieht Programmers Notepad auch ein wenig anders aus als hier auf den Screenshots.*

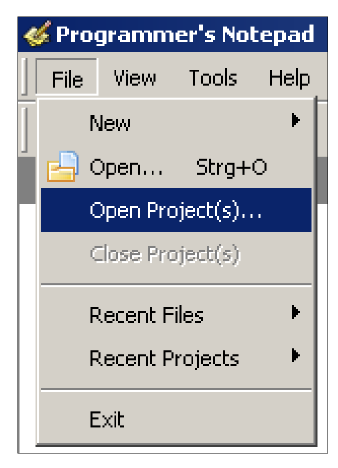
Fertig eingerichtet und nachdem man wie *im nächsten Abschnitt* beschrieben die Beispielprojekte geöffnet hat, sollte PN2 *in etwa* so aussehen:



Links sind alle Beispielprojekte zu sehen, rechts der Quelltexteditor (mit dem angesprochenen Syntax Highlighting) und unten die Ausgabe der Tools (in diesem Fall die Ausgabe des Compilers).

Sie können noch viele andere Sachen in PN2 umstellen und es gibt viele nützliche Funktionen.

4.1.3. Beispielprojekt öffnen und kompilieren



Jetzt probieren wir gleich mal aus, ob auch alles richtig funktioniert und öffnen die Beispielprojekte:

Im Menü „File" den Menüpunkt „Open Project(s)" wählen.

Es erscheint ein normaler Dateiauswahl Dialog. Hier suchen Sie bitte den Ordner „RP6Base_Examples\" im Ordner in dem Sie die Beispielprogramme gespeichert haben.

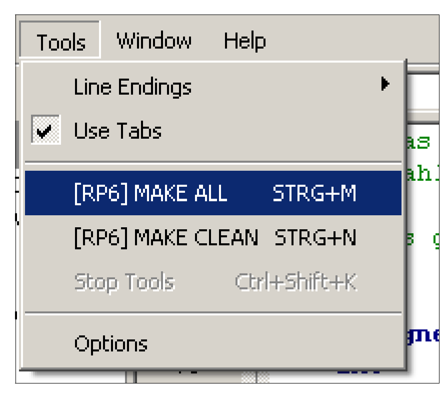
Öffnen Sie nun bitte die Datei „RP6BaseExamples.ppg". Das ist eine Projektgruppe für PN2, die alle Beispielprogramme sowie die RP6Library in die Projektliste („Projects") lädt. So hat man immer alle Beispielprogramme bequem zur Verfügung und kann sich zu Beginn besser an diesen orientieren oder Funktionen in der RP6Library nachschlagen etc..

Öffnen Sie nun das erste Beispielprogramm ganz oben in der Liste („Example_01_LEDs" und die Datei „RP6Base_LEDs.c" selektieren) die am linken Rand des Programmfensters zu sehen ist aus! Dazu einfach doppelt auf „RP6Base_LEDs.c" klicken! Es erscheint ein Quelltexteditor in einem Fenster innerhalb des Programms.

Unten im Programmfenster von PN2 sollte ein Ausgabebereich zu sehen sein – falls nicht, müssen Sie diesen Bereich im Menü mit View->Output aktivieren ODER falls er zu klein ist durch „ziehen" mit der Maus vergrößern (der Mauscursor verändert sich unten im Programmfenster am oberen Rand des grauen Bereichs in dem „Output" steht in einem recht schmalen Bereich in einen Doppelpfeil...).

Sie können sich das Programm in dem gerade geöffneten Quelltexteditor schonmal kurz anschauen, allerdings müssen Sie hier noch nicht verstehen, was da genau gemacht wird. Das wird weiter unten noch genauer erklärt. Schonmal vorweg: Bei dem grün eingefärbten Text handelt es sich um Kommentare die nicht zum eigentlichen Programm gehören und nur der Beschreibung/Dokumentation dienen. Darauf gehen wird später noch genauer ein (es gibt auch eine Version dieses Programms OHNE Kommentare – damit man mal sieht, wie kurz das Programm eigentlich ist. Die Kommentare blähen das schon ziemlich auf, sind aber zur Erklärung notwendig. Die unkommentierte Version ist auch praktisch um den Code in eigene Programme zu kopieren!).

Zunächst wollen wir nur ausprobieren, ob das Übersetzen von Programmen korrekt funktioniert.



Es sollten oben im Tools Menü die beiden eben angelegten Menüeinträge (s. Abb.) vorhanden sein (oder die standardmäßig in PN vorhandenen [WinAVR] Einträge, das ist egal, es funktioniert normalerweise mit beiden).

Klicken Sie jetzt bitte auf „MAKE ALL"!

PN2 ruft nun die oben angesprochene „make_all.bat" Batch Datei auf. Diese wiederum ruft das Programm „make" auf. Mehr zu „make" folgt noch später.

Das Beispielprogramm wird nun übersetzt (das nennt man „kompilieren“ vom englischen „to complile“ bzw. „Compiler“=„Übersetzer“) und eine Hexdatei erzeugt. Diese enthält das Programm in der für den Mikrocontroller übersetzten Form und kann dann später in diesen geladen und ausgeführt werden! Es werden während der Kompilierung noch viele temporäre Dateien erzeugt (Endungen wie „.o, .lss, .map, .sym, .elf, .dep“). Die brauchen sie *alle* nicht weiter zu beachten! Mit dem eben angelegten Tool „make clean“ kann man diese bequem löschen. Davon ist nur die Hexdatei für Sie interessant! Die Hexdatei wird beim Aufruf von „make clean“ übrigens nicht gelöscht.

Es sollte nach dem Aufruf des MAKE ALL Menüpunktes folgende Ausgabe erscheinen (hier jedoch stark gekürzt! Einiges kann natürlich etwas anders aussehen):

```
> "make" all
-------- begin --------

[...]

Compiling: RP6Base_LEDs.c

avr-gcc -c -mmcu=atmega32 -I. -gdwarf-2   -Os -funsigned-char -funsigned-bitfields -fpack-struct
-fshort-enums -Wall -Wstrict-prototypes -Wa,-adhlns=RP6Base_LEDs.lst -I../../RP6lib
-I../../RP6lib/RP6base -I../../RP6lib/RP6common -std=gnu99 -MD -MP -MF .dep/RP6Base_LEDs.o.d RP6Ba-
se_LEDs.c -o RP6Base_LEDs.o

Compiling: ../../RP6lib/RP6base/RP6RobotBaseLib.c

[...]

Creating load file for Flash: RP6Base_LEDs.hex
avr-objcopy -O ihex -R .eeprom RP6Base_LEDs.elf RP6Base_LEDs.hex

[...]


Size after:
AVR Memory Usage

----------------

Device: atmega32


Program:    6858 bytes (20.9% Full)
(.text + .data + .bootloader)

Data:        148 bytes (7.2% Full)
(.data + .bss + .noinit)


-------- end --------
> Process Exit Code: 0
> Time Taken: 00:01
```

Wichtig ist ganz unten das „`Process Exit Code: 0`“. Das bedeutet, dass es beim Übersetzen keinen Fehler gegeben hat. Steht dort ein anderer Code, gibt es einen Fehler im Quellcode, den man korrigieren muss bevor es klappt. Der Compiler gibt in diesem Fall weiter oben diverse Fehlermeldungen aus, in denen man mehr Infos dazu findet.

Aber bitte beachten Sie, dass „Process Exit Code: 0“ nicht auf ein komplett fehlerfreies Programm hinweist! Denkfehler in Ihrem Programm findet der Compiler natürlich nicht und er kann auch nicht verhindern, dass der Roboter vor die Wand fährt ;-)

WICHTIG: Weiter oben können auch noch Warnungen u.ä. stehen – diese sind oft sehr sehr hilfreich und weisen fast immer auf wichtige Probleme hin! Daher sollten diese immer beseitigt werden! PN2 hebt Warnungen und Fehler farbig hervor, so dass

man diese leicht identifizieren kann. Es wird auch die Zeilennummer angegeben, die der Compiler bemängelt. Wenn man auf diese farbig hervorgehobene Meldung klickt, springt PN2 im entsprechenden Editor direkt zu der jew. Zeile.

Auch sehr hilfreich ist die Angabe zum Schluss „AVR Memory Usage".

```
Size after:
AVR Memory Usage
----------------
Device: atmega32

Program:    6858 bytes (20.9% Full)
(.text + .data + .bootloader)

Data:        148 bytes (7.2% Full)
(.data + .bss + .noinit)
```

Das bedeutet hier, dass unser Programm 6858 Bytes groß ist und 148 Bytes RAM für statische Variablen reserviert sind (dazu kommen noch die dynamischen Bereiche für Heap und Stack, das würde an dieser Stelle aber zu weit führen... halten Sie einfach immer mindestens ein paar hundert Bytes Speicher frei). Wir haben insgesamt 32KB (32768 Bytes) an Flash ROM und 2KB (2048 Bytes) an RAM. Von den 32KB sind 2K mit dem Bootloader belegt – also können wir nur 30KB nutzen. Immer darauf achten, dass das Programm auch noch in den verfügbaren Speicher passt! (Der RP6Loader überträgt das Programm nicht wenn es zu groß ist!)

Bei dem Beispielprogramm oben sind also noch 23682 Bytes frei. Das eigentlich recht kurze Beispielprogramm RP6Base_LEDs.c ist übrigens nur deshalb schon so groß, weil die RP6Library mit eingebunden wird! Also keine Sorge, es ist genug Platz für Ihre Programme vorhanden und so kleine Programme brauchen normalerweise nicht so viel Speicher. Die Funktionsbibliothek benötigt alleine nämlich schon über 6.5KB vom Flashspeicher, nimmt Ihnen aber auch sehr viel Arbeit ab und daher werden Ihre eigenen Programme meist relativ klein sein im Vergleich zur RP6Library.

4.2. Programme in den RP6 laden

Das eben kompilierte Programm, kann nun mit dem RP6Loader in den Roboter geladen werden.

Dazu fügen Sie die eben erzeugte Hexdatei in die Liste im RP6Loader mit „Add" bzw. „Hinzufügen" ein, selektieren diese und klicken auf den „Upload!" Button, genau wie Sie es auch schon beim Selbsttestprogramm getan haben. Danach können Sie wieder auf das Terminal wechseln und sich die Ausgabe des Programms anschauen. Die Programmausführung muss dazu natürlich zunächst wieder gestartet werden, im Terminal ist es am bequemsten [STRG]+[S] auf der Tastatur zu drücken oder das Menü zu benutzen (oder einfach ein „s" senden – nach einem Reset müssen Sie allerdings immer etwas warten bis die Meldung „[READY]" im Terminal erscheint!). Auch [STRG]+[Y] ist eine sehr nützliche Tastenkombination, denn damit wird das aktuell selektierte Programm in den RP6 geladen und direkt danach gestartet! Man muss also nicht extra vom Terminal aus wieder auf den Karteireiter „Flash Loader" wechseln oder das Menü benutzen.

Das Beispielprogramm ist sehr einfach und besteht nur aus einem kleinen LED Lauflicht und etwas Textausgabe.

Bevor Sie nun Ihre eigenen Programme schreiben können, folgt ein kleiner C Crashkurs...

4.3. Warum ausgerechnet C? Und was bedeutet „GCC"?

Die Programmiersprache C ist sehr weit verbreitet – es ist die standard Sprache die eigentlich jeder der sich für Softwareentwicklung interessiert mal verwendet haben sollte (oder zumindest von der Syntax her ähnliche Sprachen). Für so gut wie jeden derzeit verfügbaren Mikrocontroller existiert mindestens ein C Compiler. Aus diesem Grund können alle neueren Roboter von AREXX Engineering (zur Zeit ASURO, YETI und der RP6) in C programmiert werden.

Da C sehr weit verbreitet ist, gibt es sehr viel Dokumentation dazu im Internet und in Büchern. Das macht es Einsteigern natürlich einfacher, auch wenn C schon eine relativ komplexe Sprache ist, die man ohne Vorkenntnisse normalerweise *nicht mal eben so* innerhalb von zwei drei Tagen erlernen kann... (also bitte nicht gleich den Roboter aus dem Fenster werfen, wenn es mal nicht auf Anhieb klappen sollte ;-))

Die Grundlagen sind zum Glück einfach zu verstehen und man kann seine Fähigkeiten kontinuierlich ausbauen und verbessern. Das erfordert aber schon etwas Eigeninitiative! Von alleine lernt sich C nicht – das ist ähnlich wie mit normalen Fremdsprachen!

Wenn man C aber erstmal einigermaßen beherrscht, ist der Einstieg in viele andere Programmiersprachen auch kein allzugroßes Problem mehr, da oft sehr ähnliche Konzepte verwendet werden.

Für den RP6 kommt wie auch bei unseren anderen Robotern eine spezielle Version des C Compilers aus der GNU Compiler Collection oder kurz GCC zum Einsatz. Es handelt sich beim GCC um ein universelles Compiler System, welches verschiedenste Sprachen unterstützt. So kann man damit z.B. neben C auch in C++, Java, Ada und FORTRAN verfasste Quelltexte übersetzen.

Der GCC unterstützt nicht nur den AVR, sondern wurde eigentlich für viel größere Systeme entwickelt und kennt einige dutzend verschiedene Zielsysteme.

Prominentestes Projekt, für das der GCC verwendet wird, ist natürlich Linux. Auch fast alle Anwendungsprogramme die unter Linux laufen, wurden ebenfalls mit dem GCC übersetzt. Es handelt sich hier also um ein sehr ausgereiftes und professionelles Werkzeug, das auch in vielen großen Firmen zum Einsatz kommt.

Übrigens: Wenn wir von „GCC" sprechen, meinen wir in diesem Handbuch nicht unbedingt die komplette Compiler Collection, sondern fast immer nur den C-Compiler. Ursprünglich stand GCC sogar nur für „GNU C Compiler" - die neuere Bedeutung wurde notwendig, als auch andere Sprachen hinzukamen.

Wenn Sie mehr über den GCC erfahren möchten, können Sie die offizielle Homepage besuchen: `http://gcc.gnu.org/`

Der GCC unterstützt den AVR nicht sofort von sich aus und muss erst angepasst werden. Diese Version des GCC nennt sich dann AVR-GCC. Dieser Compiler ist für Windows Benutzer fertig eingerichtet in WinAVR enthalten. Bei Linux muss man sich diesen meist noch selbst übersetzen, was Sie ja hoffentlich schon erledigt haben.

4.4. C - Crashkurs für Einsteiger

*Dieses Kapitel gibt Ihnen **nur eine ganz kurze Einführung** in die C Programmierung. Wir besprechen hier nur die absolut notwendigen Dinge die man für den RP6 unbedingt braucht. Dieser Abschnitt und viele der Beispielprogramme verstehen sich eher als Kurzübersicht: „Was es so alles gibt und was man alles machen kann". Es werden Beispiele gegeben und die Grundlagen erklärt, aber es wird dem Leser überlassen sich dann ausführlicher damit zu befassen!*

Es ist also nicht mehr als ein kleiner Crashkurs! Eine vollständige Einführung würde den Rahmen dieser *Bedienungsanleitung* bei weitem sprengen und füllt normalerweise dicke Fachbücher! Davon gibt es aber glücklicherweise sehr viele! Einige[1] davon sind sogar kostenlos im Internet verfügbar - hier folgt nur eine kleine Übersicht...

4.4.1. Literatur

Die folgenden Bücher und Tutorials beschäftigen sich mit der C Programmierung hauptsächlich für den PC und andere "große" Rechner. *Vieles was in diesen Büchern steht, gibt es nicht für AVR Mikrocontroller - die Sprache ist zwar gleich, aber die meisten Bibliotheken die man auf normalen PCs verwenden kann, sind für einen Mikrocontroller schlicht zu groß. Bestes Beispiel sind Funktionen wie „printf" - auf dem PC eine Selbstverständlichkeit! Diese Funktion gibt es zwar auch für Mikrocontroller, jedoch braucht sie ziemlich viel Speicher und Rechenzeit und sollte daher besser nicht verwendet werden. Wir besprechen später noch genügend für unsere Zwecke effektivere Alternativen.*

Einige C Tutorials / Online-Bücher (nur eine winzig kleine Auswahl):

`http://www.galileocomputing.de/openbook/c_von_a_bis_z/`

`http://de.wikibooks.org/wiki/C-Programmierung`

`http://suparum.rz.uni-mannheim.de/manuals/c/cde.htm`

`http://www.roboternetz.de/wissen/index.php/C-Tutorial`

`http://www.its.strath.ac.uk/courses/c/`

Weiterhin gibt es viele Bücher auf dem Markt - einfach mal in eine gut sortierte Bibliothek gehen oder bei Buchhändlern stöbern! Da findet sich meist eine ganze Menge.

Sie müssen sich kein Buch kaufen wenn Sie nur ein wenig mit dem Roboter herumexperimentieren wollen - vieles muss man sich ohnehin "Learning by doing" beibringen! Alle benötigten Informationen finden sich auch auf den genannten Seiten im Internet und die auf der CD mitgelieferten Beispielprogramme sind auch schon recht umfangreich.

Speziell für den AVR-GCC und AVR Mikrocontroller gibt es ein sehr gutes deutschsprachiges Tutorial, und zwar hier:

`http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial`

Dort werden einige Dinge genannt (Programmiergeräte etc.) die man für den RP6 aber nicht braucht. Das meiste davon ist jedoch sehr nützlich und es lohnt sich auf jeden Fall dort mal reinzuschauen!

---

1 *Eine Suche bei einer bekannten Suchmaschine nach „c tutorial" liefert viele Millionen Treffer! Es gibt natürlich nicht wirklich so viele – aber es werden trotzdem so einige sein.*

Ein weiterer Artikel, der sich aber eher mit dem Compiler an sich und einigen speziellen Bereichen befasst ist hier zu finden:

`http://www.roboternetz.de/wissen/index.php/Avr-gcc`

Dann gibt es auch noch die WinAVR Homepage und Dokumentation:

`http://winavr.sourceforge.net/`

`http://winavr.sourceforge.net/install_config_WinAVR.pdf`

sowie die AVR-LibC Dokumentation:

`http://www.nongnu.org/avr-libc/user-manual/index.html`

Sie müssen natürlich nicht alle diese Tutorials/Bücher lesen! Diese Liste hier versteht sich zur Auswahl! Aber nicht alle Tutorials sind gleichermaßen ausführlich und viele behandeln sehr unterschiedliche Themen. Es lohnt sich also schon, mehrere verschiedene zu lesen.

4.4.2. Erstes Beispielprogramm

Wie gesagt - learning by doing ist der beste Weg um die Sprache C zu lernen. Wenn Sie also etwas in diesem Crashkurs gelesen und soweit verstanden haben, sollten Sie es auch ausprobieren! Probieren geht hier wirklich über Studieren!

Natürlich müssen noch ein paar Gundlagen besprochen werden, aber damit Sie direkt einen Eindruck davon bekommen, worüber wir hier eigentlich die ganze Zeit reden, schauen wir uns jetzt mal an, wie ein kleines C Programm für den RP6 typischerweise ausschaut:

```c
/*
 * Ein kleines "Hallo Welt" C Programm für den RP6!
 */

#include "RP6RobotBaseLib.h"

int main(void)
{
      initRobotBase();
      writeString("Hallo Welt!\n");
      return 0;
}
```

Wenn Sie noch nie zuvor in C programmiert haben, sieht das wahrscheinlich auf den ersten Blick aus wie eine Fremdsprache (und das ist es ja auch!), aber die Grundlagen sind wirklich sehr einfach zu verstehen. C wurde im englischsprachigen Raum entwickelt[2] und deshalb sind auch alle Befehle an die englische Sprache angelehnt. Das ist aber nicht nur bei C so, sondern bei so gut wie allen Programmiersprachen.

2 ...genauer gesagt zu Beginn der 1970er Jahre in den USA, wo es dann als Entwicklungsgrundlage des UNIX Betriebssystems gedient hat – es folgten später noch viele Verbesserungen und Ergänzungen...

Wir halten uns bei den größeren Beispielprogrammen auch konsequent daran und benennen alle Funktionen und Variablen in den Beispielprogrammen mit englischen Begriffen! Hier im Handbuch und diesem C Crashkurs wird jedoch ab und an auch mal eine Ausnahme gemacht.

Bei den Programmen dieses Crashkurses sind die Kommentare auf Deutsch verfasst, aber bei den richtigen Beispielprogrammen und der RP6Library sind sie komplett auf Englisch.

Das hat den einfachen Grund, dass man ansonsten zwei Versionen des Quellcodes pflegen müsste und jedesmal wenn etwas geändert wird oder neue Funktionen hinzukommen, alles in zweifacher Ausführung ändern und in zwei Sprachen kommentieren müsste. Das macht nicht nur viel Arbeit, sondern verzögert auch die Freigabe von Verbesserungen und neuen Versionen.

Wer C lernen will bzw. sich detaillierter mit Mikrocontrollern, Elektronik und Robotik befasst, kommt ohnehin nur schwer um Englisch herum.

Allerdings wird die Funktionsweise der Beispielprogramme auch noch im entsprechenden Kapitel auf Deutsch erläutert!

Nun aber zurück zum Beispiel. Dieses kleine Programm oben ist zwar funktionsfähig, tut aber nichts weiter als den Mikrocontroller zu initialisieren und den Text:

`"Hallo Welt!"` + Zeilenvorschub / Neue Zeile

über die serielle Schnittstelle auszugeben! Ein typisches Programm, das in so gut wie jedem Buch zu finden ist.

Das kleine Beispielprogramm können Sie natürlich auch ausprobieren. Es kann sehr sinnvoll sein das einfach mal abzutippen um ein Gefühl für die Sprache zu bekommen! Vor allem an die vielen Semikolons und Sonderzeichen muss man sich erstmal gewöhnen...

Wem das obige Programm aber schon zu langweilig ist: Auf der CD findet sich bei den Beispielprogrammen noch ein etwas interessanteres "Hallo Welt" Programm mit kleinem LED-Lauflicht und ein paar mehr Textausgaben!

Wir werden nun das Programm in Listing 1, Zeile für Zeile durchgehen und erklären!

Zeile 1 - 3: `/* Ein kleines "Hallo Welt" C Programm für den RP6! */`
Das ist ein Kommentar. Dieser Teil des Quelltextes wird vom Compiler nicht weiter beachtet. Kommentare werden zur Dokumentation des Quelltextes verwendet und erleichtern das Verständnis fremder (und eigener!) Quelltexte. Bei fremden Quelltexten kann man damit besser nachvollziehen, was sich der Programmierer dabei gedacht hat und bei eigenen Quelltexten hat man auch nach ein paar Jahren noch gute Anhaltspunkte um sich an die eigenen Gedankengänge besser erinnern zu können. Kommentare beginnen mit /* und enden mit */

Sie können dazwischen beliebig lange Kommentare verfassen, oder auch Teile Ihres Quellcodes „auskommentieren" um z.B. eine andere Variante zu testen ohne den alten Quellcode zu löschen. Neben diesem mehrzeilen Kommentaren unterstützt der GCC auch einzeilige Kommentare, die mit einem „//" eingeleitet werden. Alles in einer Zeile nach einem „//" wird vom Compiler als Kommentar interpretiert.

Zeile 5: `#include "RP6RobotBaseLib.h"`

Hier binden wir die RP6 Funktionsbibliothek ein. Diese stellt sehr viele nützliche Funktionen und vordefinierte Dinge bereit, die das Ansteuern der Hardware immens erleichtern. Das müssen wir über eine sog. Headerdatei (Endung „*.h“) einbinden, da der Compiler sonst nicht weiss, wo er die ganzen Funktionen finden kann. Header braucht man für alle Dinge, die in externen C-Dateien liegen. Wenn Sie sich mal den Inhalt von der RP6RobotBaseLib.h und dann noch der RP6RobotBaseLib.c ansehen, werden Sie das Prinzip wahrscheinlich besser verstehen. Mehr zu „#include“ folgt im Kapitel über den Präprozessor.

Zeile 7: `int main(void)`

Das Wichtigste an dem Beispielprogramm: Die Main Funktion. Was eine Funktion genau ist, wird etwas später erklärt. Zunächst reicht es uns zu wissen, dass das Programm hier anfängt (engl. „Main Function“ bedeutet zu deutsch „Haupt-Funktion“).

Zeile 8 und Zeile 12: `{ }`

In C werden sogenannte "Blöcke" mit geschweiften Klammern - also '{' und '}' definiert! (Auf der Tastatur sind das [AltGr] + [7] für '{' und [AltGr] + [0] für '}').

Blöcke fassen mehrere Befehle zusammen.

Zeile 9: `initRobotBase();`

Hier wird eine Funktion aus der RP6Library aufgerufen, die den AVR Mikrocontroller initialisiert. Damit werden die Hardware Funktionen des AVRs konfiguriert. Ohne diesen Aufruf arbeiten die meisten Funktionen des Mikrocontrollers nicht korrekt! Also niemals vergessen das als erstes aufzurufen!

Zeile 10: `writeString("Hello World!\n");`

Hier wird die Funktion "writeString" aus der RP6 Library mit dem Text "Hello World!\n" als Parameter aufgerufen. Der Text wird dann von dieser Funktion über die serielle Schnittstelle ausgegeben.

Zeile 11: `return 0;`

Hier endet das Programm – die Main Funktion wird verlassen und der Wert 0 zurückgegeben. Das wird auf großen Computern meist für Fehlercodes oder ähnliches benutzt. Auf dem Mikrocontroller braucht man das aber eigentlich gar nicht und es ist nur da weil es der C Standard so will.

Jetzt haben Sie schonmal einen kleinen Eindruck davon bekommen wie so ein C-Programm ausschaut. Nun müssen wir aber erstmal noch ein paar Grundlagen besprechen, bevor es weitergeht.

4.4.3. C Grundlagen

Wie schon zu Beginn gesagt, besteht ein C Programm aus reinem ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchage) Text. Es wird dabei strikt zwischen Groß- und Kleinschreibung unterschieden. Das heisst sie müssen eine Funktion die "ichBinEineFunktion" heisst, auch immer genau so aufrufen! Der Aufruf: "IcHbInEiNeFuNkTIoN();" würde so nämlich nicht funktionieren!

Man kann zwischen den einzelnen Befehlen und Anweisungen beliebig viele Leerzeichen, Tabulatoren und Zeilenumbrüche einfügen ohne das sich die Bedeutung des Programms ändert.

Sie haben ja schon bei dem Beispielprogramm gesehen, dass die Befehle mit Tabulatoren eingerückt worden sind, um das Programm besser lesbar zu machen. Notwendig ist das aber nicht! Den Teil ab Zeile 7 vom Listing oben könnte man z.B. auch so schreiben:

```
int main(void){initRobotBase();writeString("Hallo Welt!\n");return 0;}
```

Das ist von der Bedeutung her identisch – aber etwas unübersichtlich. Hier wurden nur die Zeilenumbrüche und Einrückungen entfernt! Dem Compiler ist das völlig egal! (Einige Leerzeichen z.B. zwischen „int“ und „main“ sind natürlich immer notwendig um einzelne Schlüsselwörter und Bezeichner zu trennen und innerhalb von Anführungszeichen darf man auch nicht einfach so einen Zeilenumbruch einfügen!)

Mehrere Anweisungen werden mit den oben angesprochenen geschweiften Klammern { } zu sog. Blöcken zusammengefasst. Das brauchen wir bei Funktionen, Bedingungen und Schleifen.

Jede Anweisung wird mit einem Semikolon ';' beendet, so kann der Compiler die Anweisungen auseinanderhalten.

Ein wichtiger Hinweis gleich zu Beginn, wenn Sie die Programme in diesem Tutorial abtippen wollen: Man vergisst sehr leicht die Befehle jeweils mit einem Semikolon zu beenden – oder setzt eins an eine falsche Stelle und wundert sich dann, warum das Programm nicht das tut was es soll! Vergisst man ein einziges Semikolon in bestimmten Programmteilen, kann der Compiler übrigens gleich einen ganzen Haufen von Fehlermeldungen erzeugen – auch wenn es eigentlich nur ein einziger Fehler ist. Meist zeigt schon die erste Fehlermeldung den tatsächlichen Fehler an.

Auch vergisst man gern eine der vielen Klammern zu schließen oder vertut sich mit der Schreibweise von Befehlen. Der Compiler verzeiht keine Syntax Fehler (= „Rechtschreibfehler“)! Das ist vielleicht erstmal gewöhnungsbedürftig sich an all diese Regeln zu halten, aber das lernt man nachdem man einige Programme selbst geschrieben hat relativ schnell.

Jedes C Programm beginnt in der Main Funktion. Die dann folgenden Befehle werden grundsätzlich sequentiell, Anweisung nach Anweisung von oben nach unten abgearbeitet. Mehrere Befehle gleichzeitig kann ein AVR Mikrocontroller nicht ausführen!

Das macht aber nichts, denn es gibt viele Möglichkeiten, den Progammablauf zu beeinflussen und an andere Stellen im Programm zu springen (mit sog. Flusssteuerung, dazu kommen wir später).

4.4.4. Variablen

Erstmal fangen wir aber damit an, wie wir denn überhaupt Daten im Arbeitsspeicher ablegen und auslesen können. Das geht mit Variablen. Es gibt in C verschiedene Arten von Variablen - sog. Datentypen. Grundsätzlich sind dies 8, 16 oder 32Bit große, ganzzahlige Datentypen, die entweder mit oder ohne Vorzeichen verwendet werden können. Wieviel "Bit" eine Variable haben muss, hängt davon ab wie groß die Zahlen sind (Wertebereich), die darin gespeichert werden sollen.

Beim RP6 verwenden wir folgende Datentypen:

Typ	**Alternative**	**Wertebereich**	**Bemerkung**
signed char	**int8_t**	8 Bit: **-128 ... +127**	1 Byte
char	**uint8_t**	8 Bit: **0 ... 255**	'' vorzeichenlos
int	**int16_t**	16 Bit: **-32768 ... +32767**	2 Bytes
unsigned int	**uint16_t**	16 Bit: **0 ... 65535**	'' vorzeichenlos
long	**int32_t**	32 Bit: **–2147483648 ... +2147483647**	4 Bytes
unsinged long	**uint32_t**	32 Bit: **0 ... 4294967295**	'' vorzeichenlos

Da insbesondere der Datentyp „int" auf verschiedenen Plattformen nicht standardisiert ist und z.B. auf unserem Mikrocontroller 16 bit, auf PCs aber 32 bit groß ist, verwenden wir die neuere standardisierte Bezeichnung: int16_t

Diese Datentypen ist immer so aufgebaut: [u] int N _t

```
u : unsigned (kein Vorzeichen)
int : Integer (ganze Zahl)
N : Anzahl der Bits, z.B. 8, 16, 32 oder 64
_t : t wie „type" um Verwechslungen mit anderen Bezeichnern auszuschließen.
```

Auf einem Mikrocontroller kommt es oft auf jedes Byte Speicherplatz an, daher behält man mit dieser Bezeichnung auch besser den Überblick. Man sieht der Bezeichnung sofort an, dass es sich um einen 16bit Datentyp handeln muss (wegen der 16 im Namen). Steht noch ein „u" davor, handelt es sich um einen „unsigned" also vorzeichenlosen Datentyp. Sonst ist der Datentyp „signed" also vorzeichenbehaftet.

Das wurde oben in der Tabelle aber nur bei „signed char" hingeschrieben, denn int und long sind standardmäßig vorzeichenbehaftet und nur char ist standardmäßig vorzeichenlos - auch wenn man es nicht explizit hinschreibt. Das resultiert aus einer Compileroption die wir verwenden und die auch fast immer beim AVR-GCC aktiviert ist.

Bei Strings (=Engl. für Zeichenketten) wird auch weiterhin „char" genutzt da sonst wegen der Definition von uint8_t ein paar Dinge aus der C Standard Bibliothek nicht damit kompatibel wären und es ohnehin logischer ist, hierfür den Datentyp char (=Engl. für „Zeichen") zu verwenden. Das wird im Kapitel über die RP6Library bei Textausgabe über die serielle Schnittstelle noch etwas genauer behandelt.

Also merken wir uns einfach: Bei Zeichen und Zeichenketten immer „char" verwenden, bei Zahlenwerten immer uintN_t oder intN_t!

Um eine Variable nutzen zu können, muss sie zunächst deklariert werden. Hierbei wird festgelegt welchen Datentyp, Namen und evtl. noch welchen Anfangswert die Variable erhalten soll. Der Name einer Variablen muss dabei mit einem Buchstaben beginnen (Der Unterstrich also "_" zählt ebenfalls als Buchstabe) und darf auch Zahlen, aber *keine* Sonderzeichen wie z.B. „äöüß#'[]²³|*+-.,<>%&/(){}$§=´°?!^" o.ä. enthalten.

Groß- und Kleinschreibung werden wie immer unterschieden. Also sind aBc und abC verschiedene Variablen! Traditionellerweise werden immer Kleinbuchstaben zumindest für den Anfangsbuchstaben von Variablennamen verwendet.

Folgende Bezeichnungen sind bereits für andere Dinge reserviert und können NICHT als Variablennamen, Funktionsnamen oder jegliche anderen Bezeichner verwendet werden:

auto	**default**	**float**	**long**	**sizeof**	**union**
break	**do**	**for**	**register**	**static**	**unsigned**
case	**double**	**goto**	**return**	**struct**	**void**
char	**else**	**if**	**short**	**switch**	**volatile**
const	**enum**	**int**	**signed**	**typedef**	**while**
continue	**extern**				

Es gibt weiterhin noch Fließkommazahlen der Typen float und double. Man sollte auf einem kleinen AVR Mikrocontroller aber möglichst vermeiden damit zu arbeiten. Kostet meistens zuviel Rechenzeit und Speicherplatz und man kommt oft mit normalen ganzen Zahlen besser aus. Für den RP6 brauchen wir diese Datentypen also normalerweise nicht.

Eine Variable zu deklarieren ist sehr einfach, hier z.B. ein char mit dem Namen x:

```
char x;
```

Nach dieser Deklaration ist die Variable x im danach folgenden Programmcode gültig und kann verwendet werden.

Man kann ihr z.B. mit

```
x = 10;
```

den Wert 10 zuweisen. Man kann der Variablen auch gleich bei der Deklaration einen Wert zuweisen:

```
char y = 53;
```

Die normalen Grundrechenarten können ganz normal verwendet werden:

```
signed char z; // Char mit Vorzeichen!
z = x + y;   // z hat nun den Wert z = x + y = 10 + 53 = 63
z = x - y;   // z hat nun den Wert z = 10 - 53 = -43
z = 10 + 1 + 2 - 5;   // z = 8
z = 2 * x;   // z = 2 * 10 = 20
z = x / 2;   // z = 10 / 2 = 5
```

Es gibt dann noch ein paar nützliche Abkürzungen:

```
z += 10;  // entspricht: z = z + 10; also ist z = 15
z *= 2;   // z = z * 2 = 30
z -= 6;   // z = z - 6 = 24
z /= 4;   // z = z / 4 = 8
```

```
z++;  // Abkürzung für z = z + 1; Also ist z hier 9
z++;  // z = 10  // z++ nennt man auch „z inkrementieren"
z++;  // z = 11 ...
z--;  // z = 10  // z-- nennt man auch „z dekrementieren"
z--;  // z = 9
z--;  // z = 8 ...
```

Oben haben wir noch den Datentyp „char" verwendet. Wie aber schon gesagt, werden in allen anderen Programmen des RP6 (fast) nur standardisierte Datentypen verwendet.

So z.B. wäre folgendes: `int8_t x;`

identisch zu: `signed char x;`

Und: `uint8_t x;`

identisch zu: `unsigned char x;  // bzw. bei uns eigentlich auch einfach nur „char" da dieser Datentyp standardmäßig „unsigned" ist.`

4.4.5. Bedingungen

Bedingungen mit „if-else" Konstrukten sind sehr sehr wichtig um den Programmablauf zu steuern. Man kann mit ihnen überprüfen, ob eine gegebene Bedingung wahr oder falsch ist und je nachdem bestimmten Programmcode ausführen oder nicht.

Direkt ein kleines Beispiel:

```
1 uint8_t x = 10;
2 if(x == 10)
3 {
4       writeString("x ist gleich 10!\n");
5 }
```

Hier wird in Zeile 1 zunächst die 8-Bit Variable x deklariert und ihr der Wert 10 zugewiesen. Jetzt überprüfen wir in der nachfolgenden if-Bedingung in Zeile 2, ob die Variable x den Wert 10 hat. Das ist hier natürlich immer der Fall und somit wird der Block unter der Bedingung ausgeführt und „x ist gleich 10!" ausgegeben. Hätten wir x stattdessen mit 231 initialisiert, wäre an dieser Stelle nichts ausgegeben worden!

Allgemein hat eine if-Bedingung immer folgende Syntax:

```
if ( <Bedingung X> )
      <Anweisungsblock Y>
else
      <Anweisungsblock Z>
```

Ins Deutsche übersetzt heisst das übrigens soviel wie: „wenn X dann Y sonst Z".

Ein weiteres Beispiel dazu:

```
uint16_t tolleVariable = 16447;

if(tolleVariable < 16000) // Wenn tolleVariable < 16000
{                         // Dann:
	writeString("tolleVariable ist kleiner als 16000!\n");
}
else                      // Sonst:
{
	writeString("tolleVariable ist größer oder gleich 16000!\n");
}
```

Hier wäre die Ausgabe „tolleVariable ist größer oder gleich 16000!", denn tolleVariable ist hier 16447 und somit größer als 16000. Die Bedingung ist also nicht erfüllt und der else Teil wird ausgeführt. Wie man am Namen „tolleVariable" erkennen kann, hat man bei der Namensgebung von Variablen (und allem anderen) keine anderen Einschränkungen als die weiter oben genannten.

Man kann auch mehrere If-then-else-Bedingungen hintereinander verwenden um mehrere alternative Fälle abzufragen:

```
if(x == 1) { writeString("x ist 1!\n"); }
else if(x == 5) { writeString("x ist 5!\n"); }
else if(x ==  244) { writeString("x ist 244!\n"); }
else { writeString("x ist hat einen anderen Wert!\n");}
```

Innerhalb der Bedingungen kann man folgende Vergleichsoperatoren verwenden:

x == y	logischer Vergleich auf Gleichheit
x != y	logischer Vergleich auf Ungleichheit
x < y	logischer Vergleich auf „kleiner"
x <= y	logischer Vergleich auf „kleiner oder gleich"
x > y	logischer Vergleich auf „größer"
x >= y	logischer Vergleich auf „größer oder gleich"

Dann gibt es noch logische Verknüpfungsoperatoren:

x && y	wahr, wenn x wahr und y wahr sind
x || y	wahr, wenn x wahr und/oder y wahr
!x	wahr, wenn x nicht wahr ist

Das kann man alles beliebig miteinander Verknüpfen und verschachteln und beliebig viele Klammern setzen:

```
if( ((x != 0) && (!(x > 10))) || (y >= 200)) {
	writeString("OK!\n");
}
```

Die obige if Bedingung wird wahr, wenn x ungleich 0 `(x != 0)` UND x nicht größer als 10 ist `(!(x > 10))` ODER wenn y größer oder gleich 200 ist `(y >= 200)`. Dort könnte man noch beliebig viele weitere Bedingungen hinzufügen, sofern notwendig.

4.4.6. Switch-Case

Oft muss man eine Variable auf viele verschiedene Zahlenwerte überprüfen und abhängig davon Programmcode ausführen. Das kann man natürlich wie oben beschrieben mit vielen if-then-else Bedingungen machen, aber es geht auch eleganter mit einer switch-Verzweigung.

Ein Beispiel:

```
uint8_t x = 3;

switch(x)
{
	case 1: writeString("x=1\n"); break;
	case 2: writeString("x=2\n"); break;
	case 3: writeString("x=3\n");   // Hier fehlt das "break", also fährt
	case 4: writeString("Hallo\n"); // das Programm direkt mit dem
	case 5: writeString("du\n");   // nächsten und dem nächsten Fall fort,
	case 6: writeString("da!\n"); break; // und stoppt erst hier!
	case 44: writeString("x=44\n"); break;
	// Hier gelangt das Programm hin, wenn keiner der obigen
	// Fälle gepasst hat:
	default : writeString("x ist was anderes!\n"); break;
}
```

Das ist vom Resultat her *ähnlich* zu dem Beispiel auf der vorherigen Seite mit „if-else-if-else-if-else...“, nur eine andere Schreibweise.

Die Ausgabe wäre in diesem Fall (mit x = 3):

```
x=3
Hallo
du
da!
```

Wenn man x = 1 setzen würde, wäre die Ausgabe „`x=1\n`“ und mit x=5 wäre die Ausgabe:

```
du
da!
```

Hier sieht man, dass das „break“ die Switch-Verzweigung beendet. Lässt man es weg, läuft das Programm einfach solange durch alle anderen Fälle durch bis entweder das Ende der switch-Verzweigung erreicht ist oder ein anderes „break“ ausgeführt wird. Dabei ist es egal ob die nachfolgenden Bedingungen erfüllt werden oder nicht!

Wenn x = 7 gesetzt wird, würde keiner der Fälle diesen Wert abdecken, das Programm im default Teil landen und die Ausgabe wäre: „`x ist was anderes!\n`“.

Diese ganzen Textausgaben sind natürlich nur Beispiele – man könnte in realen Programmen für den Roboter damit z.B. verschiedene Bewegungsmuster auslösen. In einigen Anwendungsbeispielen werden switch-case Konstrukte z.B. für endliche Automaten verwendet um verschiedene Verhalten des Roboters zu implementieren.

4.4.7. Schleifen

Schleifen werden immer dann gebraucht, wenn bestimmte Operationen mehrmals wiederholt werden müssen.

Dazu direkt ein kleines Beispiel:

```
uint8_t i = 0;
while(i < 10)                  // solange i kleiner als 10 ist...
{                              // ... wiederhole folgenden Programmcode:
      writeString("i=");       // "i=" ausgeben,
      writeInteger(i, DEC);    // den dezimalen (engl. "DECimal") Zahlenwert
                               // von i und ...
      writeChar('\n');         // ... einen Zeilenumbruch ausgeben.
      i++;                     // i inkrementieren.
}
```

Hier handelt es sich um eine „while“ Schleife die der Reihe nach „i=0\n“, „i=1\n“, „i=2\n“, ... „i=9\n“ ausgibt. Der Block nach dem sog. Schleifenkopf, also dem „while(i < 10)“ wird solange wiederholt, wie die Bedingung wahr ist. In diesem Fall ist die Bedingung „i ist kleiner als 10“. Auf Deutsch würde das obige also soviel wie „wiederhole den folgenden Block, solange i kleiner als 10 ist“ heissen. Da i hier zunächst 0 ist und bei jedem Schleifendurchlauf um 1 erhöht wird, läuft die Schleife insgesamt 10 mal durch und gibt die Zahlen von 0 bis 9 aus. Die Bedingung im Schleifenkopf kann genau so aufgebaut werden wie bei den if Bedingungen.

Neben der while Schleife gibt es noch die „for“ Schleife. Die funktioniert ganz ähnlich, nur kann man hier noch etwas mehr im Schleifenkopf unterbringen.

Dazu wieder ein Beispiel:

```
uint8_t i;  // wird hier NICHT initialisiert, sondern im Schleifenkopf!
for(i = 0; i < 10; i++)
{
      writeString("i=");
      writeInteger(i, DEC);
      writeChar('\n');
}
```

Diese Schleife erzeugt exakt dieselbe Ausgabe wie die obige while Schleife. Hier ist nur noch einiges mehr im Schleifenkopf untergebracht.

Der grundsätzliche Aufbau ist folgender:
for (<Laufvariable initialisieren> ; <Abbruchbedingung> ; <Laufvariable verändern>)
{
        <Anweisungsblock>
}

Für Mikrocontroller braucht man oft Endlosschleifen, also Schleifen die „unendlich“ oft wiederholt werden. Fast jedes Mikrocontrollerprogramm hat eine Endlosschleife – sei es um das Programm nachdem es abgearbeitet worden ist in einen definierten Endzustand zu versetzen, oder einfach Operationen solange auszuführen wie der Mikrocontroller läuft.

Das kann man sowohl mit einer while als auch mit einer for Schleife ganz einfach realisieren:

```
while(true) { }
```

bzw.

```
for(;;) { }
```

Der Anweisungsblock wird dann „unendlich oft" ausgeführt (bzw. bis der Mikrocontroller das Reset Signal erhält, oder man mit dem Befehl „break" die Schleife beendet).

Der Vollständigkeit halber kann man noch die do-while Schleife erwähnen, dabei handelt es sich um eine Variante der normalen while-Schleife. Der Unterschied ist, dass der Anweisungsblock mindestens einmal ausgeführt wird, auch wenn die Bedingung nicht erfüllt ist.

Der Aufbau ist folgender:

```
do
{
      <Anweisungsblock>
}
while(<Bedingung>);
```

Hier das Semikolon am Ende nicht vergessen! (Bei normalen Schleifen gehört da natürlich keins hin!)

4.4.8. Funktionen

Ein sehr wichtiges Sprachelement sind Funktionen. Wir haben ja weiter oben auch schon mehrere solcher Funktionen gesehen und verwendet. Beispielsweise die Funktionen „writeString" und „writeInteger" und natürlich die Main Funktion.

Funktionen sind immer nützlich, wenn bestimmte Programmsequenzen in mehreren Teilen des Programms unverändert verwendet werden können – gute Beispiele sind z.B. die ganzen Textausgabe Funktionen die wir oben schon oft verwendet haben. Es wäre natürlich sehr umständlich wenn man diese identischen Programmteile immer an diejenigen Stellen kopieren müsste wo man sie braucht und man würde unnötigerweise Speicherplatz verschwenden. Nebenbei kann man mit Funktionen Änderungen an zentraler Stelle durchführen und muss das nicht an mehreren Stellen tun. Auch kann ein Programm deutlich übersichtlicher werden, wenn man es in mehrere Funktionen aufteilt.

Deshalb kann man in C Programmsequenzen zu Funktionen zusammenfassen. Diese haben immer folgenden Aufbau:

```
<Rückgabetyp> <Funktionsname> (<Parameter 1>, <Parameter 2>, ... <Parameter n>)
{
      <Programmsequenz>
}
```

Damit man sich das besser vorstellen kann, direkt mal ein kleines Beispiel mit zwei einfachen Funktionen und der schon bekannten Main Funktion:

```c
void someLittleFunction(void)
{
      writeString("[Funktion 1]\n");
}

void someOtherFunction(void)
{
      writeString("[Funktion 2 – mal was anderes]\n");
}

int main(void)
{
      initRobotBase(); // Beim RP6 das hier immer als erstes aufrufen!

      // Ein paar Funktionsaufrufe:
      someLittleFunction();
      someOtherFunction();
      someLittleFunction();
      someOtherFunction();
      someOtherFunction();
      return 0;
}
```

Die Ausgabe des Programms wäre folgende:

```
[Funktion 1]
[Funktion 2 – mal was anderes]
[Funktion 1]
[Funktion 2 – mal was anderes]
[Funktion 2 – mal was anderes]
```

Die Main Funktion dient als Einsprungpunkt ins Programm. Sie wird in jedem C Programm zuerst aufgerufen und MUSS somit auch immer vorhanden sein.

In unserer obigen Main Funktion wird zuerst die initRobotBase Funktion aus der RP6Library aufgerufen, die den Mikrocontroller initialisiert (muss man beim RP6 immer als erstes in der Main Funktion aufrufen, sonst funktionieren viele Sachen nicht richtig!). Diese ist vom Prinzip her genau so aufgebaut wie die anderen beiden Funktionen im Listing. Anschließend werden nacheinander ein paar mal die beiden gerade definierten Funktionen aufgerufen und der Programmcode in den Funktionen ausgeführt.

Nun kann man Funktionen nicht nur wie oben im Beispiel gezeigt definieren, sondern auch noch Parameter und Rückgabewerte benutzten. Im obigen Beispielprogramm wird als Parameter und Rückgabetyp „void" angegeben, was soviel wie „leer" heisst. Das bedeutet einfach, dass diese Funktionen keinen Rückgabewert und keine Parameter haben. Man kann viele Parameter für eine Funktion definieren. Die Parameter werden dabei durch Kommata voneinander getrennt.

Ein Beispiel:

```
void outputSomething(uint8_t something)
{
    writeString("[Der Funktion wurde folgender Wert übergeben: ");
    writeInteger(something, DEC);
    writeString("]\n");
}

uint8_t calculate(uint8_t param1, uint8_t param2)
{
    writeString("[CALC]\n");
    return (param1 + param2);
}

int main(void)
{
    initRobotBase();

    // Ein paar Funktionsaufrufe:
    outputSomething(199);
    outputSomething(10);
    outputSomething(255);

    uint8_t result = calculate(10, 30);
    outputSomething(result);
    return 0;
}
```

Ausgabe:

```
[Der Funktion wurde folgender Wert übergeben: 199]
[Der Funktion wurde folgender Wert übergeben: 10]
[Der Funktion wurde folgender Wert übergeben: 255]
[CALC]
[Der Funktion wurde folgender Wert übergeben: 40]
```

Die RP6 Library enthält ebenfalls sehr viele verschiedene Funktionen. Schauen Sie sich einfach mal einige davon und die Beispielprogramme an, dann wird das Prinzip sehr schnell klar.

4.4.9. Arrays, Zeichenketten, Zeiger ...

Es gäbe noch viel mehr Sprachelemente und Details zu besprechen, aber hier müssen wir auf die angegebene Literatur verweisen. (Infos zu Zeigern finden Sie z.B. hier: http://www.galileocomputing.de/openbook/c_von_a_bis_z/c_014_000.htm)

Das meiste davon braucht man auch gar nicht um die Beispielprogramme verstehen zu können. Hier geben wir nur ein paar Beispiele und Begriffe an, um einen Überblick darüber zu geben. Eine sonderlich ausführliche Beschreibung ist das allerdings nicht.

Zunächst zu Arrays. In einem Array (="Feld") kann man eine vorgegebene Anzahl von Elementen eines bestimmten Datentyps speichern. So könnte man z.B. folgendermaßen ein Array mit 10 Bytes anlegen:

```
uint8_t unserTollesArray[10];
```

Schon hat man quasi 10 Variablen mit gleichem Datentyp deklariert, die man nun über einen Index ansprechen kann:

```
unserTollesArray[0] = 8;
unserTollesArray[2] = 234;
unserTollesArray[9] = 45;
```

Jedes dieser Elemente kann man wie eine normale Variable behandeln.

Achtung: Der Index startet immer bei 0 und wenn man ein Array mit n Elementen angelegt hat, geht der Index von 0 bis n-1 ! Also bei 10 Elementen z.B. von 0 bis 9.

Arrays ist sehr nützlich wenn man viele Werte gleichen Typs speichern muss. Man kann diese auch in einer Schleife der Reihe nach durchlaufen:

```
uint8_t i;
for(i = 0; i < 10; i++)
      writeInteger(unserTollesArray[i],DEC);
```

So werden der Reihe nach alle Elemente im Array ausgegeben (hier natürlich ohne Trennzeichen oder Zeilenumbrüche dazwischen). Analog dazu kann man auch ein Array in einer Schleife mit Werten beschreiben.

Ganz ähnlich sind Zeichenketten in C aufgebaut. Normale Zeichenketten sind stets im ASCII Code kodiert, wofür man pro Zeichen ein Byte benötigt. Zeichenketten sind in C nichts anderes als Arrays, die man eben als Zeichenketten interpretieren kann. Man kann z.B. folgendes Schreiben:

```
uint8_t eineZeichenkette[16] = "abcdefghijklmno";
```

Und schon hat man die in Klammern stehende Zeichenkette im Speicher abgelegt.

Wir haben weiter oben auch schon ein paar UART Funktionen verwendet, die Zeichenketten über die serielle Schnittstelle ausgeben. Die Zeichenketten sind dabei nur Arrays. Der Funktion wird allerdings kein komplettes Array übergeben, sondern nur die Adresse des ersten Elements in dem Array. Die Variable die diese Adresse enthält nennt man „Zeiger" (engl. Pointer). Um einen Zeiger auf ein bestimmtes Array Element zu erzeugen schreibt man &unserTollesArray[x] wobei x das Element ist, auf das der Zeiger zeigen soll. Diese Schreibweise wird ab und an in den Beispielprogrammen verwendet. Beispielsweise so:

```
uint8_t * zeigerAufEinElement =  &eineZeichenkette[4];
```

Das brauchen Sie aber erstmal noch nicht um die Programmbeispiele zu verstehen und eigene Programme zu schreiben. Wird hier nur der Vollständigkeit halber erwähnt.

4.4.10. Programmablauf und Interrupts

Wir hatten weiter oben schon gesagt, das ein Programm generell Anweisung nach Anweisung von oben nach unten ausgeführt wird. Dann gibt es natürlich noch die normale Flusssteuerung mit Bedingungen und Schleifen, sowie Funktionen.

Neben diesem normalen Programmablauf gibt es allerdings noch sog. „Interrupts“. Die verschiedenen Hardwaremodule (Timer, TWI, UART, Externe Interrupts etc.) des Mikrocontrollers können Ereignisse auslösen, auf die der Mikrocontroller so schnell wie möglich reagieren muss. Dazu springt der Mikrocontroller – (fast) egal wo er sich gerade im normalen Programm befindet – in eine sog. Interrupt Service Routine (ISR), in der dann schnellstmöglich auf das Ereignis reagiert werden kann. Keine Sorge, Sie müssen sich hier nicht noch selbst drum kümmern, für alle benötigten ISRs wurde das bereits in der RP6Library erledigt. Aber damit Sie wissen worum es geht und was diese seltsamen „Funktionen“ in der Library sollen, erwähnen wir es hier kurz.

Die ISRs sehen wie folgt aus:

```
ISR ( <InterruptVector> )
{
	<Anweisungsblock>
}
```

z.B. für den linken Encoder am externen Interrupt Eingang 0:

```
ISR (INT0_vect)
{
	// Hier werden bei jeder Signalflanke zwei Zähler erhöht:
	mleft_dist++;      // zurückgelegte Distanz
	mleft_counter++;  // Geschwindigkeitsmessung
}
```

Man kann diese ISRs nicht direkt aufrufen! Das geschieht immer automatisch und meist kann man nicht vorhersagen wann es genau passiert! Es kann zu jeder Zeit in jedem beliebigen Programmteil passieren (ausser in einem Interrupt selbst oder wenn Interrupts deaktiviert sind). Dann wird die ISR ausgeführt und danach wieder an die Stelle zurückgesprungen an der das Programm unterbrochen wurde. Daher muss man, wenn man Interrupts einsetzt, alle zeitkritischen Dinge auch in den jeweiligen Interrupt Service Routinen erledigen. Pausen die man anhand von Taktzyklen errechnet hat, können sonst zu lang werden wenn sie von einem oder mehreren Interrupt Ereignissen unterbrochen werden.

In der RP6Library werden Interrupts verwendet um die 36kHz Modulation für die Infrarotsensorik und Kommunikation zu erzeugen. Außerdem für die RC5 Dekodierung, Timing und Delay Funktionen, um die Encoder auszuwerten, für das TWI Modul (I²C-Bus) und noch ein paar andere kleinere Dinge.

4.4.11. C-Präprozessor

Den C Präprozessor wollen wir auch noch kurz ansprechen. Wir haben diesen bereits oben verwendet – nämlich bei `#include "RP6RobotBaseLib.h"`!

Dieser Befehl wird noch vor dem eigentlichen Übersetzen durch den GCC vom Präprozessor ausgewertet. Mit `#include "Datei"` wird der Inhalt der angegebenen Datei an dieser Stelle eingefügt. Im Falle unseres Beispielprogramms ist das die Datei RP6BaseLibrary.h. Diese enthält Definitionen der in der RP6Library enthaltenen Funktionen. Das muss so gemacht werden damit der Compiler diese Funktionen auch findet und richtig zuordnen kann.

Aber das ist natürlich noch nicht alles was man mit dem Präprozessor machen kann. Man kann auch Konstanten (also feste nicht veränderbare Werte) definieren:

```
#define DAS_IST_EINE_KONSTANTE 123
```

Das würde die Konstante: „`DAS_IST_EINE_KONSTANTE`" mit dem Wert „123" definieren. Der Präprozessor ersetzt einfach jedes Vorkommen dieser Konstanten durch den Wert (eigentlich ist es Textersatz). Also würde z.B. hier:

```
writeInteger(DAS_IST_EINE_KONSTANTE,DEC);
```

das „`DAS_IST_EINE_KONSTANTE`" durch „123" ersetzt und wäre somit identisch zu:

```
writeInteger(123,DEC);
```

(übrigens ist auch das „DEC" von writeInteger nur eine so definierte Konstante – hier mit dem Wert 10 – für das dezimale Zahlensystem.)

Auch einfache If-Bedingungen kennt der Präprozessor:

```
1 #define DEBUG
2
3 void someFunction(void)
4 {
5       // Mach irgendwas
6       #ifdef DEBUG
7          writeString_P("someFunction wurde ausgeführt!");
8       #endif
9 }
```

Der Text würde hier nur ausgegeben wenn DEBUG definiert wurde (es muss kein Wert zugewiesen werden – einfach nur definieren). Nützlich um z.B. diverse Ausgaben die man nur für die Fehlersuche braucht nur bei Bedarf zu aktivieren. Wenn DEBUG im obigen Beispiel nicht definiert wäre, würde der Präprozessor den Inhalt von Zeile 7 gar nicht an den Compiler weiterleiten.

In der RP6Library benötigen wir oft auch Makros – diese werden ebenfalls per #define definiert, man kann ihnen aber Parameter übergeben, ähnlich wie bei Funktionen.

z.B. so:

```
#define setStopwatch1(VALUE) stopwatches.watch1 = (VALUE)
```

Das lässt sich dann wie eine normale Funktion aufrufen (`setStopwatch1(100);`).

Eine wichtige Sache ist noch, dass man normalerweise hinter Präprozessor Definitionen <u>kein</u> Semikolon schreibt!

4.5. Makefiles

Das Tool „Make“ nimmt uns eine ganze Menge Arbeit ab, denn normalerweise müsste man so einiges in die Kommandozeile eintippen um ein C Programm zu kompilieren.

Make verarbeitet ein sog. „Makefile“, das alle Befehlssequenzen und Informationen enthält, die zum Kompilieren des jeweiligen Projekts notwendig sind. Diese Makefiles werden bei jedem der RP6 Beispielprojekte schon fertig mitgeliefert – man kann natürlich später auch eigene Makefiles erstellen. Wie so ein Makefile im Detail aufgebaut ist, kann hier nicht beschrieben werden – das wäre zu umfangreich. Sie brauchen sich aber für alle RP6 Projekte ohnehin nur um die folgenden vier Einträge Gedanken zu machen – der Rest ist für Einsteiger erstmal uninteressant:

```
TARGET = programmName

RP6_LIB_PATH=../../RP6lib

RP6_LIB_PATH_OTHERS=$(RP6_LIB_PATH)/RP6base $(RP6_LIB_PATH)/RP6common

SRC += $(RP6_LIB_PATH)/RP6base/RP6RobotBaseLib.c
SRC += $(RP6_LIB_PATH)/RP6common/RP6uart.c
SRC += $(RP6_LIB_PATH)/RP6common/RP6I2CslaveTWI.c
```

In unseren Makefiles sind dazwischen noch einige Kommentarzeilen mit Erläuterungen und Hinweisen zu finden. Kommentare fangen in Makefiles immer mit „#“ an und werden von Make nicht weiter beachtet.

Die Beispielprojekte für den RP6 enthalten bereits fertige Makefiles mit den passenden Einträgen. Sie müssen diese also nur dann ändern, wenn Sie z.B. neue C Dateien zum Projekt hinzufügen wollen, oder Dateien umbenennen möchten.

Beim Eintrag „TARGET“ trägt man den Namen der C Datei ein, die die Main-Funktion enthält. Hier gibt man nur den Namen an, OHNE das „.c“ am Ende! Bei vielen anderen Einträgen muss diese Endung aber angegeben werden, also immer auf die gegebenen Beispiele und Hinweise in den Kommentaren achten!

Im Eintrag „RP6_LIB_PATH“ müssen Sie das Verzeichnis mit den Dateien der RP6Library angeben. Das ist i.d.R. „../RP6lib“ oder „../../RP6lib“ also ein relativer Pfad. („../“ bedeutet „eine Verzeichnisebene höher“)

Bei RP6_LIB_PATH_OTHERS muss man alle sonstigen Verzeichnisse angeben die man verwendet hat. Die RP6Library ist in mehrere Verzeichnisse unterteilt, von denen man alle die man verwendet auch angeben muss.

Und schließlich bei „SRC“ müssen Sie alle C-Dateien angeben (keine Header Dateien! Also nicht die Dateien die mit „.h“ enden! Die werden automatisch in allen angegebenen Verzeichnissen gesucht!), die Sie neben der Datei mit der Main-Funktion verwenden wollen. Auch die Dateien aus der RP6Library müssen hier angegeben werden, sofern diese verwendet werden sollen.

Was bedeutet eigentlich $(RP6_LIB_PATH) ? Ganz einfach: So kann man in Makefiles Variablen verweden! Wir haben ja schon eine „Variable“ RP6_LIB_PATH definiert. Den Inhalt kann man nun mit $(<Variable>) überall nach dieser Deklaration einfügen. Ist praktisch und erspart sehr viel Tipparbeit...

Mehr brauchen Sie an den Makefiles für den RP6 normalerweise nicht zu ändern. Wenn Sie aber mehr wissen möchten, finden Sie hier ein ausführliches Handbuch: http://www.gnu.org/software/make/manual/

4.6. RP6 Funktionsbibliothek (RP6Library)

Die RP6 Funktionsbibliothek, auch RP6Library oder kurz RP6Lib genannt, bietet viele nützliche Funktionen um die Hardware des RP6 anzusteuern. Über die meisten hardwarespezifischen Dinge muss man sich damit (eigentlich) keine Gedanken mehr machen. Sie müssen also natürlich nicht das über 300 Seiten starke Datenblatt des ATMEGA32 gelesen haben um den Roboter programmieren zu können! Es ist aber sehr nützlich, wenn man trotzdem grundlegend nachvollzieht, was die RP6Library Funktionen machen. Außerdem sind viele Funktionen in der RP6Library absichtlich nicht ganz perfekt – so bleibt für Sie auch noch etwas zu tun!

Man könnte noch so einige Zusatzfunktionen hinzufügen und vieles optimieren! Die Funktionen der RP6Library verstehen sich als Grundlage, auf der man aber sehr gut aufbauen kann.

In diesem Abschnitt beschreiben wir die wichtigsten Funktionen und geben kleine Beispiele. Wer weitere Informationen dazu benötigt, sollte die Kommentare in den Dateien der Library lesen und sich die Funktionen und Beispiele genau ansehen und nachvollziehen.

4.6.1. Mikrocontroller initialisieren

```
void initRobotBase(void)
```

Diese Funktion müssen Sie IMMER als erstes in der Main Funktion aufrufen!

Sie initialisiert die Hardwaremodule des Mikrocontrollers. Nur wenn Sie diese Funktion als erstes aufrufen, wird der Mikrocontroller so arbeiten, wie wir es für den RP6 benötigen! Ein Teil wird zwar schon vom Bootloader initialisiert, aber nicht alles.

Beispiel:

```
1  #include "RP6RobotBaseLib.h"
2
3  int main(void)
4  {
5        initRobotBase(); // Initialisierung - IMMER ALS ERSTES AUFRUFEN!
6
7  // [...] Programmcode...
8
9        while(true);     // Endlosschleife
10       return 0;
11 }
12
```

Jedes RP6 Programm muss <u>mindestens</u> so ausschauen! Die Endlosschleife in Zeile 9 ist notwendig, um ein definiertes Ende des Programms zu garantieren! Ohne diese Schleife könnte sich das Programm anders verhalten als erwartet!

Nur um das zu verdeutlichen: Normalerweise führt man in der Endlosschleife eigenen Programmcode aus, also würde man hier in Zeile 9 das Semikolon löschen und stattdessen einen Block (zwei geschweifte Klammern) einfügen in den das eigene Programm kommt. Vor der Main Funktion (also Zeile 2) kann man eigene Funktionen definieren, die man dann aus der Hauptschleife beliebig oft aufrufen kann.

4.6.2. UART Funktionen (serielle Schnittstelle)

Wir haben im C-Crashkurs oben schon einige Funktionen aus der RP6Library verwendet, vor allem die UART Funktionen. Mit diesen Funktionen kann man Textnachrichten über die serielle Schnittstelle des Roboters an den PC (oder andere Mikrocontroller) schicken und auch von diesem emfangen.

4.6.2.1. Senden von Daten über die serielle Schnittstelle

void writeChar(char ch)

Diese Funktion schickt ein einzelnes 8-Bit ASCII Zeichen über die serielle Schnittstelle. Die Anwendung ist sehr einfach:

```
writeChar('A');
writeChar('B');
writeChar('C');
```

Gibt „ABC“ aus. Man kann auch direkt ASCII Codes übertragen:

```
writeChar(65);
writeChar(66);
writeChar(67);
```

Das ergibt im Terminal ebenfalls die Ausgabe „ABC“ - jedes ASCII Zeichen ist schließlich einer bestimmten Nummer zugeordnet, das 'A' z.B. der 65. Mit einer angepassten Kommunikationssoftware, könnte man aber auch die reinen binären Werte interpretieren.

Oft braucht man auch:

```
writeChar('\n');
```

um eine neue Zeile im Terminal anzufangen.

void writeString(char *string)
und **writeString_P(STRING)**

Diese Funktionen sind sehr wichtig für die Fehlersuche in Programmen, denn hiermit kann man beliebige Textnachrichten an den PC schicken. Für Datenübertragungen kann man sie natürlich ebenso verwenden.

Der Unterschied zwischen writeString und writeString**_P** besteht darin, dass bei Verwendung von writeString_P die Texte nur im Flash-ROM (**P**rogram Memory) abgelegt werden und auch von dort gelesen werden, bei writeString jedoch zusätzlich in den Arbeitsspeicher geladen werden und somit doppelt Speicherplatz verbraucht wird. Und vom Arbeitsspeicher haben wir nunmal nur 2KB zur Verfügung. Wenn es nur um die Ausgabe von festem und nicht veränderlichem Text geht, sollte man deshalb immer writeString_P verwenden. Wenn man dynamische Daten ausgeben will, die ohnehin im RAM vorliegen, **muss** man natürlich das normale writeString verwenden.

Auch hier ist die Anwendung denkbar einfach:

```
writeString("ABCDEFG");
```

Gibt „ABCDEFG“ aus, belegt aber für die Zeichenkette auch Arbeitsspeicher.

```
writeString_P("ABCDEFG");
```

Gibt ebenfalls „ABCDEFG“ aus, aber ohne dabei unnötig RAM zu belegen!

```
void writeStringLength(char *data, uint8_t length, uint8_t offset);
```

Wenn man Texte mit einstellbarer Länge (length) und Anfangsposition (offset) ausgeben möchte, kann man diese Funktion verwenden.

Ein Beispiel:

```
writeStringLength("ABCDEFG", 3, 1);
```

Ausgabe: „BCD"

```
writeStringLength("ABCDEFG", 2, 4);
```

Ausgabe: „EF"

Diese Funktion belegt allerdings auch RAM Speicher und ist eigentlich nur für dynamische Textausgabe gedacht (wird z.B. von writeIntegerLength verwendet...).

```
void writeInteger(int16_t number, uint8_t base);
```

Diese sehr nützliche Funktion gibt Zahlenwerte als ASCII Text aus! Wir haben ja oben schon gesehen, dass z.B. writeChar(65) ein 'A' ausgibt und nicht 65...

Daher braucht man so eine Konvertierungsfunktion.

Beispiel:

```
writeInteger(139, DEC);
```

Ausgabe: „139"

```
writeInteger(25532, DEC);
```

Ausgabe: „25532"

Man kann den gesamten 16bit Wertebereich mit Vorzeichen von -32768 bis 32767 ausgeben. Wer größere Zahlen braucht, muss die Funktion anpassen oder besser eine eigene Funktion schreiben!

Was soll das „DEC" als zweiter Parameter? Ganz einfach: Das bedeutet, das die Ausgabe als Dezimalzahl (engl. DECimal) erfolgt. Es gibt aber noch andere Zahlensysteme als das Dezimalsystem mit Basis 10. So kann man die Werte auch binär (BIN, Basis 2), oktal (OCT, Basis 8) oder hexadezimal (HEX, Basis 16) ausgeben.

Beispiele:

```
writeInteger(255, DEC);
```

Ausgabe: „255"

```
writeInteger(255, HEX);
```

Ausgabe: „FF"

```
writeInteger(255, OCT);
```

Ausgabe: „377"

```
writeInteger(255, BIN);
```

Ausgabe: „11111111"

Das kann für viele Anwendungen sehr nützlich sein – vor allem HEX und BIN, denn hier sieht man sofort ohne Kopfrechnen wie die einzelnen Bits in dem Zahlenwert angeordnet sind.

```
void writeIntegerLength(uint16_t number, uint8_t base, uint8_t length);
```

Eine Variante von writeInteger. Hier kann zusätzlich die Anzahl der Stellen (length) die ausgegeben werden soll, angegeben werden. Ist eine Zahl kürzer als die angegebene Stellenzahl, werden führende Nullen angefügt. Ist Sie länger, werden nur die letzten Stellen dargestellt.

Beispiele:

```
writeIntegerLength(2340, DEC, 5);
```

Ausgabe: „02340"

```
writeIntegerLength(2340, DEC, 8);
```

Ausgabe: „00002340"

```
writeIntegerLength(2340, DEC, 2);
```

Ausgabe: „40"

```
writeIntegerLength(254, BIN, 12);
```

Ausgabe: „000011111110"

4.6.2.2. Empfangen von Daten über die serielle Schnittstelle

Diese Funktionen wurden in Version 1.3 der RP6Library komplett neu geschrieben und daher wurde dieser Abschnitt daran angepasst.

Der Empfang von Daten über die serielle Schnittstelle läuft nun komplett Interrupt basiert ab. Die empfangenen Daten werden automatisch im Hintergrund in einem sog. Ringpuffer gespeichert.

Einzelne empfangene Bytes/Zeichen kann man mit der Funktion

```
char readChar(void)
```

aus dem Ringpuffer lesen. Ruft man diese Funktion auf, wird das jeweils nächste verfügbare Zeichen zurückgegeben und aus dem Ringpuffer gelöscht.

Ist der Ringpuffer leer, gibt die Funktion 0 zurück. Allerdings sollte man vor jedem Aufruf mit der Funktion

```
uint8_t getBufferLength(void)
```

überprüfen wieviele neue Zeichen noch im Ringpuffer vorhanden sind.

Mehrere Zeichen hintereinander können mit der Funktion

```
uint8_t readChars(char *buf, uint8_t numberOfChars)
```

aus dem Puffer abgerufen werden. Als Parameter übergibt man der Funktion einen Zeiger auf ein Array und die Anzahl der Zeichen die in dieses Array kopiert werden sollen. Die Funktion gibt die tatsächliche Anzahl kopierter Zeichen zurück. Das ist nützlich falls nicht soviele Zeichen wie angefordert im Puffer vorhanden waren.

Sollte der Puffer komplett voll sein, werden alte Daten beim Empfang von neuen Zeichen nicht überschrieben, sondern es wird eine status Variable gesetzt (uart_status) die einen „übergelaufenen" Puffer signalisiert (engl. „Buffer overflow", UART_BUFFER_OVERFLOW). Sie sollten Ihre Programme so auslegen, dass es nicht dazu kommt. Meist ist in diesem Fall die Datenrate zu hoch oder das Programm wurde für längere Zeit mit mSleep o.ä. blockiert. Sie können auch die Größe des Ringpuffers

anpassen. Standardmäßig hat der Ringpuffer eine Größe von 32 Zeichen. In der RP6uart.h Datei können Sie dies mit der UART_RECEIVE_BUFFER_SIZE Definition anpassen.

Ein größeres Programm dazu finden Sie bei den Beispielprogrammen! Das Beispiel „Example_02_UART_02“ wurde an die neuen Funktionen angepasst.

4.6.3. Delay Funktionen (Verzögerungen und Zeitsteuerung)

Oft muss man Verzögerungen (Delays) in sein Programm einbauen, oder eine bestimmte Zeitspanne warten bevor eine bestimmte Aktion ausgeführt wird.

Auch dafür hat die RP6Library Funktionen. Für die Verzögerungen einer der Timer des MEGA32 verwendet, um möglichst unabhängig von der sonstigen Programmausführung (Interrupts) zu sein und die Verzögerungen einigermaßen genau zu halten.

Aber man muss genau überlegen wann man diese Funktionen verwenden kann! Wenn man die automatische Geschwindigkeitsregelung und das ACS verwendet (wird später noch erläutert) könnte das sonst zu Problemen führen! Dann sollte man nur ganz ganz kurze Pausen mit wenigen Millisekunden (<10ms) einfügen! Das Problem kann aber leicht umgangen werden wenn man die Stopwatches verwendet die direkt im Anschluss beschrieben werden.

void sleep(uint8_t time)

Mit dieser Funktion hält man den normalen Programmablauf für eine gewisse Zeit an. Die Zeitspanne wird dabei in Schritten von 100µs übergeben (100µs = 0.1ms = 0.0001s also für die menschliche Wahrnehmung sehr kurz...). Da wir eine 8 bit Variable benutzten, ist die maximale Zeitspanne 25500µs = 25.5ms. Interrupt Ereignisse werden trotzdem weiter behandelt – es wird nur der normale Programmablauf unterbrochen.

Beispiele:

```
sleep(1); // 100µs Pause
sleep(10); // 1ms Pause
sleep(255); // 25.5ms Pause
```

void mSleep(uint16_t time)

Wenn man längere Pausen benötigt, kann man mSleep verwenden, denn hier wird die Zeitspanne in Millisekunden angegeben. Die maximale Zeitspanne beträgt 65535ms, oder 65.5 Sekunden.

Beispiele:

```
mSleep(1);     // 1ms Pause
mSleep(100);   // 100ms Pause
mSleep(1000);  // 1000ms = 1s Pause
mSleep(65535); // 65.5 Sekunden Pause
```

Stopwatches

Das Problem bei diesen normalen Delay Funktionen ist - wie oben schon gesagt - dass sie den normalen Programmablauf komplett unterbrechen. Das ist aber nicht immer erwünscht, denn oft muss nur ein Teil des Programms für eine bestimmte Zeit warten, während andere Dinge weiterlaufen sollen...

Das ist der größte Vorteil daran, Hardwaretimer zu verwenden, denn diese laufen unabhängig vom restlichen Programmablauf. Die RP6Library implementiert mit den Ti-

mern universell verwendebare „Stopwatches“. Diese Bezeichnung ist nicht allgemein üblich, der Autor fand den Namen aber passend, weil das ganz ähnlich wie reale Stoppuhren funktioniert. Die Stopwatches machen viele Aufgaben einfacher. Normalerweise schreibt man solche Timing Funktionen speziell auf das jeweilige Problem zugeschnitten, die RP6Library bietet das hier etwas universeller und einfacher an, so dass man es für viele verschiedene Dinge gleichermaßen benutzen kann.

Mit den Stopwatches kann man viele Aufgaben quasi simultan erledigen – zumindest wirkt es für den Betrachter von außen so.

Es stehen acht 16bit Stopwatches (Stopwatch1 bis Stopwatch8) zur Verfügung. Diese kann man starten, stoppen, setzen und auslesen. Die Auflösung beträgt eine Millisekunde, wie bei der mSleep Funktion. Das bedeutet, dass jede Stopwatch ihren Zählstand jede Millisekunde um 1 erhöht. Für sehr zeitkritische Dinge ist das aber nicht geeignet, da die Abfragen ob eine Stopwatch einen bestimmten Wert erreicht hat, meist nicht exakt zu diesem Zeitpunkt erfolgen.

Wie man diese Funktionen anwendet, sieht man gut an folgendem kleinen Beispielprogramm:

```
#include "RP6RobotBaseLib.h"

int main(void)
{
	initRobotBase(); // Mikrocontroller initialisieren
	writeString_P("\nRP6 Stopwatches Demo Programm\n");
	writeString_P("___________________________\n\n");

	startStopwatch1(); // Stopwatch1 starten!
	startStopwatch2(); // Stopwatch2 starten!

	uint8_t counter = 0;
	uint8_t runningLight = 1;

	// Hauptschleife:
	while(true)
	{
		// Ein kleines LED Lauflicht:
		if(getStopwatch1() > 100) // Sind 100ms (= 0.1s) vergangen?
		{
		    setLEDs(runningLight); // LEDs setzen
		    runningLight <<= 1;    // Nächste LED (shift Operation)
		    if(runningLight > 32)  // Letzte LED?
		        runningLight = 1;  // Ja, also wieder von vorn beginnen!
		    setStopwatch1(0);      // Stopwatch1 auf 0 zurücksetzen
		}

		// Einen Zählerstand auf dem Terminal ausgeben:
		if(getStopwatch2() > 1000) // Sind 1000ms (= 1s) vergangen?
		{
		    writeString_P("CNT:");
		    writeInteger(counter, DEC); // Zählerstand ausgeben
		    writeChar('\n');
		    counter++;          // Zähler um 1 erhöhen
		    setStopwatch2(0);   // Stopwatch2 auf 0 zurücksetzen
		}
	}
	return 0;
}
```

Das Programm ist sehr einfach gehalten. Es wird jede Sekunde ein Zählerstand über die serielle Schnittstelle ausgegeben und inkrementiert (Zeile 29 bis 36) und nebenbei noch ein Lauflicht ausgeführt (Zeile 19 bis 26), das alle 100ms weitergeschaltet wird. Dazu werden Stopwatch1 und Stopwatch2 verwendet. In Zeile 9 und 10 werden die beiden Stopwatches gestartet und fangen danach an zu zählen. In der Endlosschleife (Zeile 16 bis 37) wird dann ständig abgefragt, ob die Stopwatches einen bestimmten Zählstand überschritten haben. In Zeile 19 beispielsweise für das Lauflicht, hier wird abgefragt ob bereits *mindestens* 100ms vergangen sind, seit die Stopwatch von 0 an zu Zählen begonnen hat. Ist das der Fall, wird die nächste LED angeschaltet und die Stopwatch auf 0 zurückgesetzt (Zeile 25) und es wird erneut 100ms lang gewartet. Genauso wird bei dem Zähler verfahren, nur ist das Intervall hier 1000ms also eine Sekunde.

Das Programm finden Sie etwas ausführlicher auch auf der CD. Es ist nur ein kleines Beispiel, man kann auch viel komplexere Dinge mit den Stopwatches realisieren und diese z.B. auch bedingt starten und stoppen etc...

Beim Beispielprogramm auf der CD sind deshalb das Lauflicht und der Zähler (dort sind es sogar 3 Stück...) auch jeweils in einer eigenen Funktion untergebracht, die aus der Endlosschleife aufgerufen werden. Das macht es bei komplexeren Programmen übersichtlicher und man kann viele Programmteile so später in anderen Programmen einfacher per Copy&Paste wiederverwenden – wie z.B. so ein Lauflicht.

Es stehen mehrere Makros zur Steuerung der Stopwatches zur Verfügung.

startStopwatchX()

Startet eine bestimmte Stopwatch. Die Stopwatch wird nicht zurückgesetzt, sondern läuft einfach vom letzten Zählstand aus weiter.

Beispiele:

```
startStopwatch1();
startStopwatch2();
```

stopStopwatchX()

Hält eine bestimmte Stopwatch an.

Beispiel:

```
stopStopwatch2();
stopStopwatch1();
```

uint8_t isStopwatchXRunning()

Gibt zurück ob die Stopwatch X läuft.

Beispiel:

```
if(!isStopwatch2Running) {
	// Stopwatch läuft nicht, also mache irgendwas...
}
```

setStopwatchX(uint16_t preset)

Dieses Makro setzt Stopwatch X auf den übergebenen Wert.

Beispiel:

```
setStopwatch1(2324);
setStopwatch2(0);
setStopwatch3(2);
setStopwatch4(43456);
```

getStopwatchX()

Dieses Makro gibt den Wert von Stopwatch X zurück.

Beispiel:

```
if(getStopwatch2() > 1000) { ... }
if(getStopwatch6() > 12324) { ... }
```

4.6.4. Status LEDs und Bumper

void setLEDs(uint8_t leds)

Die 6 Status LEDs können Sie mit dieser Funktion steuern. Am übersichtlichsten ist es, der Funktion einen binären Wert zu übergeben (sieht immer so aus: 0bxxxxxx).

Beispiel:

```
setLEDs(0b000000); // Dieser Befehl schaltet alle LEDs aus.
setLEDs(0b000001); // Und dieser schaltet StatusLED1 an und alle anderen aus.
setLEDs(0b000010); // StatusLED2
setLEDs(0b000100); // StatusLED3
setLEDs(0b001010); // StatusLED4 und StatusLED2
setLEDs(0b010111); // StatusLED5, StatusLED3, StatusLED2 und StatusLED1
setLEDs(0b100000); // StatusLED6
```

Eine alternative Möglichkeit ist folgendes:

```
statusLEDs.LED5 = true;   // LED5 im LED Register aktivieren
statusLEDs.LED2 = false;  // LED2 im LED Register deaktivieren
updateStatusLEDs();       // Änderungen übernehmen!
```

Hier wird die StatusLED5 angeschaltet und StatusLED2 ausgeschaltet. Die anderen LEDs bleiben aber genau in dem Zustand in dem sie vorher gewesen sind! Das kann nützlich/übersichtlicher sein, wenn verschiedene LEDs aus verschiedenen Funktionen heraus geändert werden sollen o.ä..

Achtung: `statusLEDs.LED5 = true;` schaltet NICHT direkt LED5 an! Es wird nur ein Bit in einer Variablen gesetzt! Erst `updateStatusLEDs();` schaltet LED5 dann wirklich an!

Zwei der Portpins an denen die LEDs angeschlossen sind, werden zusätzlich für die Bumper verwendet. Um die Bumper auszuwerten wird für sehr kurze Zeit der jeweilige Portpin auf Eingangsrichtung umgeschaltet und ausgewertet ob der Schalter geschlossen ist. Dazu gibt es zwei vorgefertigte Funktionen.

Diese Funktion

uint8_t getBumperLeft(void)

wertet den linken Bumper aus und diese

uint8_t getBumperRight(void)

den rechten Bumper.

Da diese Funktionen sehr schnell ausgeführt werden, werden die LEDs nicht dunkel auch wenn die Funktion den Portpin auf Eingangsrichtung umschaltet. Man sollte aber zwischen den einzelnen Aufrufen ein paar Millisekunden Pause einfügen.

Die Ports sollten nur über die vorgefertigten Funktionen angesteuert werden! Die Ports an denen die Bumper angeschlossen sind, sind zwar durch Widerstände gesichert, aber wenn sie auf Masse geschaltet werden und dann ein Bumper geschlossen wird, fließt schon ein recht hoher Strom durch den Port. Das sollte also vermieden werden.

Beispiel:

```
if(getBumperLeft() && getBumperRight())  // Beide Bumper...
      escape(); // Hier eigene Funktion definieren, z.B. zurücksetzen + drehen
else if(getBumperLeft()) // Links...
      escapeLeft(); // hier ebenfalls zurücksetzen und nach rechts drehen.
else if(getBumperRight()) // Rechts...
      escapeRight(); // und hier zurücksetzen und nach links drehen.
mSleep(50); // Die Bumper nur 20 mal pro Sekunde (20Hz) auswerten...
```

Die beiden LEDs 6 und 3 leuchten in jedem Fall wenn die Bumper gedrückt werden. Das ist absichtlich so (geht nicht anders) und keineswegs eine Fehlfunktion. Normalerweise schalten die Bumper aber nur selten, deshalb stört das nicht weiter.

Sie können an die Ports der anderen vier LEDs auch weitere Bumper/Sensoren mit digitalem Ausgang oder auch Transistoren zum Schalten von Lasten, wie weiteren LEDs oder kleinen Motoren anschließen. Dazu sind allerdings noch keine passenden Funktionen implementiert...

Achtung: *Fügen Sie auf jeden Fall mindestens 470 Ohm Widerstände zwischen Sensoren/Aktoren und Ports ein, um eine Beschädigung der Ports des Mikrocontrollers durch Überstrom zu vermeiden!*

Mit dem RP6Loader kann man übrigens die LEDs während des Bootvorgangs deaktivieren. Das ist nützlich, wenn man die Ports mit irgend etwas anderem belegen will und nicht möchte, dass diese Ports beim Booten an und ausgeschaltet werden. Das erste Byte im internen EEPROM (also Adresse 0) ist für Einstellungen wie diese reserviert! Sie sollten dieses Byte also nicht in Ihren eigenen Programmen verwenden! (das stört zwar sonst nichts wichtiges, aber man könnte sich wundern warum die LEDs plötzlich nicht mehr leuchten wenn man den RP6 anschaltet...)

Es gibt auf dem RP6 viele Dinge, die ständig ausgewertet und überwacht werden müssen damit alles korrekt funktioniert. So z.B. das ACS. Hier müssen oft IR Impulse ausgesandt und Empfangen werden etc. pp.. Das kann man nicht alles automatisch in Interrupt Routinen erledigen, weil diese immer möglichst schnell ausgeführt werden müssen. Deshalb muss man aus dem Hauptprogramm ständig einige Funktionen aufrufen, die Aufgaben wie diese erledigen. Wenn man das eigene Programm richtig aufbaut, wirkt das aber so, als wenn das alles im Hintergrund laufen würde!

Diese Funktionen werden im Laufe dieses Kapitels alle noch vorgestellt. Um die Funktionen für die Bumper zu verstehen, mussten wir das hier aber schon kurz vorwegnehmen!

Wo man nämlich ohnehin schon so einiges im Hintergrund ausführt, kann man auch

gleich noch kleinere Aufgaben - wie eben die Bumper auslesen - in einem Rutsch mit erledigen. Das geht sehr schnell und man müsste das sonst ohnehin fast immer noch im Hauptprogramm machen.

Damit die Bumper automatisch überwacht werden, muss man diese Funktion:

```
void task_Bumpers(void)
```

ständig aus der Hauptschleife aufrufen (s.a. Antriebs Funktionen – hier wird das vorgehen noch ausführlicher besprochen) – sie prüft die Bumper dann automatisch alle 50ms und schreibt die Werte (gedrückt oder nicht gedrückt) in die Variablen

```
bumper_left und bumper_right
```

Diese kann man dann überall sonst im Programm ganz normal in if Bedingungen etc. verwenden und abfragen, oder anderen Variablen zuweisen.

Beispiel:

```
#include "RP6RobotBaseLib.h"

int main(void)
{
	initRobotBase();  // Mikrocontroller initialisieren

	setLEDs(0b001001); // LEDs 1 und 4 (beide Grün) anschalten

	while(true)
	{
		// Hier setzen wir die LEDs je nachdem welcher Bumper
		// gedrückt ist:
		statusLEDs.LED6 = bumper_left;  // Bumper links gedrückt
		statusLEDs.LED4 = !bumper_left; // links nicht gedrückt
		statusLEDs.LED3 = bumper_right; // Bumper rechts gedrückt
		statusLEDs.LED1 = !bumper_right; // rechts nicht gedrückt
		// Beide Bumper gedrückt:
		statusLEDs.LED2 = (bumper_left && bumper_right);
		statusLEDs.LED5 = statusLEDs.LED2;
		updateStatusLEDs(); // LEDs aktualisieren...

		// Bumper Status prüfen:
		task_Bumpers(); // ständig aus der Hauptschleife aufrufen!
	}
	return 0;
}
```

Im Beispielprogramm benutzen wir die Status LEDs um den Zustand der beiden Bumper anzuzeigen. Ist der linke Bumper gedrückt, wird LED 6 angeschaltet und LED4 abgeschaltet. Ist er nicht gedrückt, wird umgekehrt LED6 ausgeschaltet und LED4 angeschaltet. LED6 leuchtet zwar sowieso wenn der linke Bumper gedrückt wird – aber es könnte ja auch irgendetwas anderes mit den LEDs visualisiert werden, das soll nur ein Beispiel sein wie man das generell macht. Beim rechten Bumper wird dasselbe mit LED3 und LED1 gemacht. Sind beide Bumper gedrückt, wird LED2 und LED5 angeschaltet.

Da man nun sowieso schon die automatische Auswertung der Bumper alle 50ms in der Library hatte, war es naheliegend etwas einzubauen, das eine selbstdefinierte Funktion automatisch bei Zustandsänderungen der Bumper aufruft. Normalerweise werden die Bumper ja relativ selten getroffen und da ist es sinnvoll im Hauptprogramm nur

dann etwas abzufragen, wenn auch Bedarf dafür besteht.

In C kann man auch Zeiger auf Funktionen definieren und diese dann darüber aufrufen ohne die Funktion in der Library selbst zu definieren. Normalerweise müsste eine Funktion schon zur Zeit der Übersetzung des Programms bekannt sein und in der RP6-Library eingetragen werden, damit man diese Aufrufen kann.

So kann man eigens definierte Funktionen als sog. „Event Handler", also für die Ereignisbehandlung verwenden. Wird ein Bumper gedrückt, wird innerhalb von etwa 50ms automatisch eine Funktion aufgerufen, die man extra dafür erstellt und zuvor als Event Handler registriert hat. Die Funktion muss dabei eine bestimmte Signatur haben. In diesem Fall muss es eine Funktion sein, die keinen Rückgabewert und keinen Parameter hat (beides void).
Die Funktionssignatur muss also so `void bumpersStateChanged(void)` aussehen. Den Event Handler kann man beispielsweise zu Beginn der Main Funktion registrieren. Zum Registrieren dieses Event Handlers verwendet man folgende Funktion:

```
void BUMPERS_setStateChangedHandler(void (*bumperHandler)(void))
```

Die Notation müssen Sie nicht unbedingt genau verstehen – kurz gesagt wird hier ein Zeiger auf eine Funktion übergeben...
Hier gleich ein einfaches Beispielprogramm dazu:

```
#include "RP6RobotBaseLib.h"

// Unsere „Event Handler" Funktion für die Bumper.
// Die Funktion wird automatisch aus der RP6Library aufgerufen:
void bumpersStateChanged(void)
{
	writeString_P("\nBumper Status hat sich geaendert:\n");

	if(bumper_left)
		writeString_P(" - Linker Bumper gedrueckt!\n");
	else
		writeString_P(" - Linker Bumper nicht gedrueckt.\n");
	if(bumper_right)
		writeString_P(" - Rechter Bumper gedrueckt!\n");
	else
		writeString_P(" - Rechter Bumper nicht gedrueckt.\n");
}

int main(void)
{
	initRobotBase();

	// Event Handler registrieren:
	BUMPERS_setStateChangedHandler(bumpersStateChanged);

	while(true)
	{
		task_Bumpers(); // Bumper automatisch alle 50ms auswerten
	}
	return 0;
}
```

Das Programm gibt bei jeder Änderung des Bumperzustands einmal den aktuellen Status beider Bumper aus.

Drückt man z.B. den rechten Bumper wäre die Ausgabe:

```
Bumper Status hat sich geaendert:
 - Linker Bumper nicht gedrueckt.
 - Rechter Bumper gedrueckt!
```

Drückt man beide:

```
Bumper Status hat sich geaendert:
 - Linker Bumper gedrueckt!
 - Rechter Bumper gedrueckt!
```

Da man beide Bumper nicht wirklich gleichzeitig drücken kann, könnte evtl. noch eine zusätzliche Nachricht ausgegeben werden wo nur einer der beiden gedrückt gewesen ist.

Wohlgemerkt wird in dem obigen Programmcode, nirgendwo die Funktion `bumpersStateChanged` direkt aufgerufen! Das passiert automatisch aus der RP6Library heraus, nämlich aus der Funktion `task_Bumpers` bei jeder Änderung des Bumperzustands. Da `task_Bumpers` unsere Funktion eigentlich erstmal gar nicht kennt, geht das nur über einen Zeiger auf die entsprechende Funktion, der seinen Wert erst bei Ausführung des Aufrufs in Zeile 24 erhält.

Der Event Handler kann natürlich noch andere Aktionen auslösen, als nur Text auszugeben – z.B. könnte man den Roboter anhalten und zurücksetzen lassen. Das sollte man allerdings nicht direkt im Event Handler selbst tun, sondern in anderen Programmteilen – z.B. im Event Handler einfach irgendeine Kommando-Variable setzen die man im Hauptprogramm abfragt und dann dementsprechend die Motoren steuert! Alle Event Handler sollten immer so kurz wie möglich gehalten werden!

Sie können in den Event Handlern zwar alle Funktionen aus der RP6Library verwenden, seien Sie aber vorsichtig mit den „rotate" und „move" Funktionen die wir später noch besprechen werden! Benutzen Sie hier NICHT den blockierenden Modus (wenn man dann nämlich z.B. öfters auf die Bumper drückt, funktioniert das nicht ganz so wie man es eigentlich haben wollte ;-))!

Dieses Prinzip mit den Event Handlern wird auch noch für andere Funktionen verwendet, z.B. für das ACS – da ist es sogar sehr ähnlich zu den Bumpern, denn auch dort wird bei jeder Zustandsänderung der Objektsensoren ein Event Handler aufgerufen.

Auch für den Empfang von RC5 Codes über eine Fernbedienung werden Event Handler verwendet – jedesmal wenn ein neuer RC5 Code empfangen wurde, kann ein entsprechender Event Handler aufgerufen werden.

Man *muss* für all diese Dinge übrigens keine Event Handler verwenden – man kann das alles auch durch einfache if Bedingungen o.ä. abfragen und dementsprechend darauf reagieren, aber mit Event Handlern werden einige Dinge einfacher und bequemer. Ist aber eher Geschmackssache was man verwendet.

Auf der CD finden Sie übrigens noch einige ausführlichere Beispielprogramme zu diesem Thema!

4.6.5. ADC auslesen (Batterie-, Motorstrom- und Lichtsensoren)

Am ADC (Analog to Digital Convertor) sind - wie in Abschnitt 2 schon beschrieben wurde - viele Sensoren des RP6 angeschlossen. Natürlich bietet die RP6Library auch hier eine Funktion, um diese auszulesen:

```
uint16_t readADC(uint8_t channel)
```

Die Funktion gibt einen 10 Bit Wert (0...1023) zurück, also benötigt man 16 Bit Variablen für die Sensorwerte.

Es stehen folgende Kanäle zur Verfügung:

```
ADC_BAT           --> Batteriespannungs Sensor
ADC_MCURRENT_R    --> Motorstromsensor des rechten Motors
ADC_MCURRENT_L    --> Motorstromsensor des linken Motors
ADC_LS_L          --> Linker Lichtsensor
ADC_LS_R          --> Rechter Lichtsensor
ADC_ADC0          --> Freier ADC Kanal für eigene Sensoren
ADC_ADC1          --> Freier ADC Kanal für eigene Sensoren
```

Hinweis: Die Anschlüsse für die zwei freien ADC Kanäle sind nicht bestückt. Hier kann man eigene Stecker im 2.54mm Rastermaß anlöten und evtl. noch zwei kleine 100nF Kondensatoren und einen großen 470µF Elko hinzufügen falls man Sensoren anschließen möchte die hohe Spitzenströme benötigen wie z.B. IR Abstandssensoren von Sharp ... Sie sollten dazu allerdings schon ein wenig Löterfahrung haben! Sonst lieber ein Erweiterungsmodul verwenden!

Beispiele:

```
uint16_t ubat = readADC(ADC_BAT);
uint16_t iMotorR = readADC(ADC_MCURRENT_R);
uint16_t iMotorL = readADC(ADC_MCURRENT_L);
uint16_t lsL = readADC(ADC_LS_L);
uint16_t lsR = readADC(ADC_LS_R);
uint16_t adc0 = readADC(ADC_ADC0);
uint16_t adc1 = readADC(ADC_ADC1);

if(ubat < 580) writeString_P("Warnung! Batterie fast leer!");
```

Standardmäßig wird die 5V Versorgungsspannung als Referenz verwendet. Man kann aber die Funktion auch so umschreiben, dass die interne 2.56V Referenz des ATMEGA32 verwendet wird (dazu im Datenblatt vom MEGA32 nachlesen). Für die normalen Sensoren des RP6 wird das allerdings normalerweise nicht benötigt.

Es ist sinnvoll mehrere ADC Werte zu messen (z.B. in einem Array speichern) und dann zunächst einmal den Mittelwert davon zu bilden oder Minimum/Maximum zu bestimmen bevor man die Messwerte des ADCs auswertet. Ab und zu fängt man sich nämlich durch Störungen Messfehler ein. Im Falle der Akkuspannung ist es z.B. für einen automatischen Stromsparmodus absolut notwendig, ein paar Werte zu mitteln denn die Spannung kann sehr stark schwanken - vor allem wenn die Motoren laufen und wechselnd belastet werden.

Wie bei den Bumpern kann man die ADC Messungen automatisieren. Es gibt auch hier schon eine kleine Funktion, die uns das Leben etwas erleichtern kann.

```
void task_ADC(void)
```

In diesem Fall verkürzt sie die Zeit die zum Auslesen aller Analog/Digital Wandler Kanäle im Programm benötigt wird. Ruft man diese Funktion ständig auf, werden automatisch alle ADC Kanäle der Reihe nach „im Hintergrund“ (also immer wenn gerade Zeit dazu ist) ausgelesen und die Messwerte in Variablen gespeichert.

Der ADC benötigt für jede Messung etwas Zeit und mit der readADC Funktion wäre der normale Programmablauf dann für diese Zeit unterbrochen. Da die Messung aber auch ohne unser zutun läuft (der ADC ist schließlich ein Hardwaremodul im Mikrocontroller) können wir in dieser Zeit auch andere Dinge erledigen.

Es gibt für die einzelnen Kanäle folgende 16Bit Variablen, die man immer und überall im Programm verwenden kann:

```
ADC_BAT:           adcBat
ADC_MCURRENT_L:    adcMotorCurrentLeft
ADC_MCURRENT_R:    adcMotorCurrentRight
ADC_LS_L:          adcLSL
ADC_LS_R:          adcLSR
ADC_ADC0:          adc0
ADC_ADC1:          adc1
```

Sobald Sie die task_ADC() Funktion verwenden, sollten Sie nur noch diese Variablen benutzen und NICHT die readADC Funktion!

Beispiel:

```
#include "RP6RobotBaseLib.h"

int main(void)
{
	initRobotBase();
	startStopwatch1();
	writeString_P("\n\nKleines ADC Messprogramm...\n\n");
	while(true)
	{
		if(getStopwatch1() > 300) // Alle 300ms...
		{
			writeString_P("\nADC Lichtsensor Links: ");
			writeInteger(adcLSL, DEC);
			writeString_P("\nADC Lichtsensor Rechts: ");
			writeInteger(adcLSL, DEC);
			writeString_P("\nADC Akku: ");
			writeInteger(adcBat, DEC);
			writeChar('\n');
			if(adcBat < 600)
				writeString_P("Warnung! Akku ist bald leer!\n");
			setStopwatch1(0); // Stopwatch1 auf 0 zurücksetzen
		}
		task_ADC(); // ADC Auswertung - ständig aus der Hauptschleife
	}               // aufrufen! Dann sollte man aber readADC nicht
	return 0;       // mehr verwenden!
}
```

Hier werden alle 300ms die Messwerte der beiden Lichtsensoren und vom Akku ausgegeben. Unterschreitet die Akkuspannung *etwa* 6V, wird eine Warnung ausgegeben.

4.6.6. ACS – Anti Collision System

Das Anti Collision System wird nicht mehr wie beim RP5, von einem kleineren Co-Prozessor gesteuert, sondern direkt vom MEGA32. Das erfordert nun zwar etwas mehr Programmieraufwand auf dem Hauptcontroller, hat aber auch den Vorteil, dass man es beliebig verändern und verbessern kann. Beim RP5 war das Programm im Co-Prozessor nicht veränderbar...

Die Reichweite bzw. die Sendeleistung der beiden IR LEDs des ACS, kann man mit folgenden Funktionen einstellen:

```
void setACSPwrOff(void)  --> ACS IR LEDs aus
void setACSPwrLow(void)  --> Geringe Reichweite
void setACSPwrMed(void)  --> Mittlere Reichweite
void setACSPwrHigh(void) --> Maximale Reichweite
```

Wenn man das ACS auswerten möchte, muss man ständig aus der Hauptschleife heraus die Funktion:

```
void task_ACS(void)
```

aufrufen, die das ACS komplett steuert. Der Rest sieht dann ganz ähnlich wie bei den Bumpern aus.

Es gibt zwei Variablen:

obstacle_left und **obstacle_right**

welche jeweils den Wert true haben, sobald ein Hindernis detektiert wurde. Sind beide true, befindet sich das Hindernis mittig vor dem Roboter.

Wenn man möchte, kann man auch hier einen Event Handler erstellen, muss man aber nicht – also genau wie bei den Bumpern.

```
void ACS_setStateChangedHandler(void (*acsHandler)(void))
```

Mit dieser Funktion registriert man den Event Handler, der folgende Signatur haben muss: `void acsStateChanged(void)`

Wie die Funktion genau heisst, ist dabei übrigens fast egal.

Das folgende Beispielprogramm zeigt die Anwendung. Zuerst wird der Event Handler registriert (Zeile 44), dann die Stromversorgung zum IR Empfänger eingeschaltet (Zeile 46 – ohne das funktioniert es nicht!) und die Sendestärke der ACS IR LEDs eingestellt (Zeile 47). Unten in der Hauptschleife wird dann die Funktion task_ACS() ständig aufgerufen.

Der Rest erfolgt automatisch – die acsStateChanged Funktion wird immer aufgerufen, wenn sich der Zustand des ACS verändert hat – also wenn z.B. ein Objekt von den Sensoren entdeckt wird und wieder verschwindet. Das Programm stellt den aktuellen Zustand des ACS als Text im Terminal und mit den LEDs dar.

```c
#include "RP6RobotBaseLib.h"

void acsStateChanged(void)
{
	writeString_P("ACS status hat sich geändert!   L: ");

	if(obstacle_left)  // Hindernis links
		writeChar('o');
	else
		writeChar(' ');

	writeString_P(" | R: ");

	if(obstacle_right) // Hindernis rechts
		writeChar('o');
	else
		writeChar(' ');

	if(obstacle_left && obstacle_right) // Mitte?
		writeString_P("   MITTE!");
	writeChar('\n');

	statusLEDs.LED6 = obstacle_left && obstacle_right; // Mitte?
	statusLEDs.LED3 = statusLEDs.LED6;
	statusLEDs.LED5 = obstacle_left;      // Hindernis links
	statusLEDs.LED4 = (!obstacle_left);   // LED5 invertiert!
	statusLEDs.LED2 = obstacle_right;     // Hindernis rechts
	statusLEDs.LED1 = (!obstacle_right);  // LED2 invertiert!
	updateStatusLEDs();
}

int main(void)
{
	initRobotBase();

	writeString_P("\nRP6 ACS - Testprogramm\n");
	writeString_P("_____________________\n\n");

	setLEDs(0b111111);
	mSleep(1000);
	setLEDs(0b001001);

	// ACS Event Handler registrieren:
	ACS_setStateChangedHandler(acsStateChanged);

	powerON(); // ACS Empfänger einschalten (und Encoder etc.)
	setACSPwrMed(); // ACS auf mittlere Sendeleistung stellen.

	while(true)
	{
		task_ACS(); // ständig die task_ACS Funktion aufrufen!
	}
	return 0;
}
```

In diesem Beispiel sieht man auch nochmal sehr schön wie man die LEDs einzeln setzen kann.

Wenn man das Programm ausführt, sollte man den RP6 natürlich am Rechner ange-

schlossen lassen und sich die Ausgaben auf dem Bildschirm ansehen. Bewegen Sie einfach mal die Hand oder einen Gegenstand direkt vor dem Roboter hin und her!

Das ACS kann durch Störquellen in seiner Funktion beeinträchtigt werden! Bestimmte Leuchtstoffröhren und ähnliches können den Roboter quasi „blind“ machen oder zumindest die Empfindlichkeit verringern. Wenn Sie also Probleme damit haben, schalten Sie erstmal alle evtl. störenden Lichtquellen aus. (Tipp: Wenn Sie den Roboter direkt vor einem Flachbildschirm oder Flachbildfernseher platziert haben, kann es auch daran liegen. Meist wird dort schließlich auch eine Leuchtstoffröhre für die Hintergrundbeleuchtung verwendet...)
Die Reichweite hängt außerdem stark von den Gegenständen selbst ab, denn eine schwarze Oberfläche reflektiert das IR Licht natürlich weniger gut als eine helle weisse Oberfläche. Sehr dunkle Objekte werden vom ACS nicht immer erkannt!

So gesehen, sollte man das ACS noch mit Ultraschallsensoren oder besseren IR Sensoren unterstützen.

Was Sie unbedingt noch tun sollten, bevor Sie den Roboter herumfahren lassen: Einfach mal diverse Gegenstände mit dem ACS testen und schauen welche NICHT sonderlich gut erkannt werden. Dann könnte man solche Objekte evtl. woanders hinstellen während man den Roboter herumfahren lässt... ist allerdings nicht mehr ganz so wichtig wie beim Vorgänger, denn notfalls gibt es jetzt noch die Stoßstange die weitestgehend verhindert, dass die IR LEDs verbiegen!

4.6.7. IRCOMM und RC5 Funktionen

Mit dem IR Empfänger kann der RP6 auch IR Signale von normalen TV/Hifi Fernbedienungen empfangen. Das klappt allerdings nur, wenn die Fernbedienung im RC5 Code sendet! Die meisten Universalfernbedienungen (s. Abb.) können auf dieses Format eingestellt werden – wie das geht steht im Handbuch der jeweiligen Universalfernbedienung. Wenn in der Codetabelle nicht explizit der RC5 Code aufgeführt wird, kann man einfach einige verschiedene Gerätehersteller durchprobieren.

Wenn die Fernbedienung im RC5 Format sendet, wird dieses Signal von der ACS Hinderniserkennung ignoriert und das ACS nur sehr wenig bis gar nicht gestört. Es kann trotzdem noch Hindernisse detektieren, möglicherweise aber etwas später da so nur die Pausen zwischen den RC5 Übertragungen genutzt werden können. Sollte die Fernbedienug allerdings nicht im RC5 Format senden, wird natürlich der gesendete Code nicht erkannt und das ACS könnte dadurch gestört werden.

Mit einer passenden Software im Mikrocontroller, kann man den RP6 dann mit einer Fernbedienung steuern und ihm Befehle senden!

Über das IRCOMM System kann man auch IR Signale senden. Die beiden Sendedioden vorne am Roboter sind nach oben zur Decke gerichtet. Über die Reflektion an der

Zimmerdecke und anderen Gegenständen oder bei direkter Sichtverbindung nach vorn, kann man so mit anderen Robotern oder z.B. einer Basisstation kommunizieren. Das geht zwar nur relativ langsam (ein Datenpaket braucht ca. 20ms und danach eine kleine Pause), aber einfache Kommandos und einzelne Messwerte lassen sich schon damit übertragen. Die Reichweite ist natürlich etwas eingeschränkt und es funktioniert nur im selben Raum wenn die Roboter maximal etwa 2 bis 4 Meter voneinander entfernt sind (je nach Lichtverhältnissen, Hindernissen, Beschaffenheit der Zimmerdecke und den Aufbauten/Erweiterungen auf den Robotern). Die Reichweite lässt sich aber steigern, wenn man z.B. noch ein paar weitere IR LEDs hinzuschaltet (z.B. mit einem weiteren MOSFET, einem großen Kondensator und kleinem Vorwiderstand).

Aufgrund der notwendigen Synchronisation mit dem ACS ist auch für diese Aufgaben die task_ACS() Funktion zuständig. Diese muss also auch deshalb ständig in der Hauptschleife aufgerufen werden, damit auf empfangene IR Daten reagiert werden kann - und außerdem um die Übertragungen mit dem IRCOMM auszuführen!

RC5 Datenpakete enthalten immer eine Geräteadresse, den „Keycode" und ein „Togglebit". Die 5 Bit Geräteadresse gibt an, welches Gerät angesteuert werden soll. Bei normaler Verwendung kann man so die Fernbedienungen für TV, Videorecorder, HiFi Anlage etc. unterscheiden. Bei unserer Anwendung kann man anstattdessen z.B. verschiedene Roboter adressieren. Der 6 Bit Keycode ist bei normalen Fernbedienungen der Tastencode, aber wir können damit natürlich auch Daten übertragen. Zwar nur 6 Bit, aber 8 Bit Daten könnte man auf zwei Übertragungen aufteilen, oder noch 2 der 5 Bit Geräteadresse oder das Togglebit zweckentfremden.

Das Togglebit wird normalerweise dazu verwendet, um zu unterscheiden ob eine Taste auf der Fernbedienug dauerhaft gedrückt wird oder mehrmals hintereinander. Das Togglebit kann bei Datenübertragungen von Roboter zu Roboter aber beliebig verwendet werden.

RC5 Daten kann man mit dieser Funktion senden:

```
void IRCOMM_sendRC5(uint8_t adr, uint8_t data)
```

Wobei `adr` die Geräteadresse und `data` Keycode bzw. Daten sind. In `adr` kann man zusätzlich das Toggle Bit setzen, indem man das höchstwertige Bit dieses Bytes setzt. Dazu kann man die Konstante `TOGGLEBIT` wie folgt verwenden:

```
IRCOMM_sendRC5(12 | TOGGLEBIT, 40);
```

Dieser Befehl sendet ein RC5 Datenpaket an Adresse 12, bei gesetztem Togglebit und mit 40 als Daten.

```
IRCOMM_sendRC5(12, 40);
```

So sieht es dann aus, wenn man das Togglebit NICHT setzen will.

Der Empfang von RC5 Daten funktioniert über einen Event Handler, ähnlich wie bei den Bumpern und beim ACS. Der Event Handler wird automatisch aus der task_ACS() Funktion aufgerufen sobald neue RC5 Daten vorliegen. Beispielweise könnte man dann, wenn Tastencode 4 empfangen wurde, nach links drehen und bei Code 6 nach rechts o.ä....

In einem der Beispielprogramme wird genau das gemacht: man kann die Bewegung des Roboters komplett über eine IR Fernbedienung steuern.

Der Event Handler muss folgende Signatur haben:

```
void receiveRC5Data(RC5data_t rc5data)
```

Wie die Funktion heisst, ist natürlich auch hier fast egal!

```
void IRCOMM_setRC5DataReadyHandler(void (*rc5Handler)(RC5data_t))
```

Mit dieser Funktion kann man den zuvor definierten Event Handler registrieren.

z.B. so:

```
IRCOMM_setRC5DataReadyHandler(receiveRC5Data);
```

Danach wird immer diese Funktion aufgerufen sobald ein gültiger RC5 Code empfangen worden ist.

`RC5data_t` ist ein eigens definierter Datentyp, der RC5 Adressbits (Device Address), Togglebit und Keycode (bzw. die Daten) enthält. Diese sind dann ähnlich wie normale Variablen verwendbar und haben folgende Bezeichnungen:

```
rc5data.device, rc5data.toggle_bit, rc5data.key_code
```

Auf der CD finden Sie ein größeres Beispielprogramm dazu.

Achtung: schalten Sie niemals den Pin des IRCOMM dauerhaft an! Die IR LEDs und die Steuerschaltung sind für Impulsbetrieb ausgelegt und dürfen nur eine Millisekunde am Stück eingeschaltet bleiben! Die Stromaufnahme ist sonst bei voll aufgeladenen Akkus zu hoch. Ändern Sie die IRCOMM Funktionen nicht selbst ab, wenn Sie nicht genau wissen was Sie tun. Vor allem die Interrupt Routine die zur Ansteuerung verwendet wird, sollte nicht verändert werden!

4.6.8. Stromsparfunktionen

Weiter oben haben wir powerON() verwendet, ohne näher darauf einzugehen. Man kann beim RP6 das ACS, die Encoder, die Motorstromsensoren und die Power ON LED abschalten um Energie zu sparen (sind immerhin etwa 10mA wenn man diese Sensoren gerade nicht braucht).

Dazu gibt es die Makros:

```
powerON()
```

um die genannten Dinge einzuschalten und

```
powerOFF()
```

um das alles abzuschalten. Diese beiden Makros tun nichts weiter, als einen I/O Pin auf high oder low zu schalten.

Bevor man ACS, IRCOMM und Motorregelung verwenden kann, muss man in jedem Fall das Makro powerON() aufrufen! Sonst haben die entsprechenden Sensoren keine Energie. Die Motorregelung kann z.B. nur korrekt arbeiten, wenn ein Signal von den Drehgebern und Stromsensoren kommt.
Wenn Sie mal vergessen powerON() aufzurufen, werden die Motoren nach einem kurzen Startversuch sofort wieder abgeschaltet und die vier roten Status LEDs fangen an zu blinken.

4.6.9. Antriebs Funktionen

Der Antrieb des Roboters kann mit Funktionen der RP6Library sehr komfortabel gesteuert werden. Es gibt dort bereits Funktionen, die mithilfe der Drehgeber automatisch die Geschwindigkeit der Motoren einregeln, den Motorstrom überwachen, automatisch bestimmte Distanzen abfahren und noch einiges mehr. Das ist sehr komfortabel, aber man muss – wie bei den anderen Systemen auch - ein paar Besonderheiten beachten wenn man das benutzen will.

Und die aktuelle Fassung ist sicherlich noch nicht optimal. Da kann noch viel verbessert und hinzugefügt werden!

```
void task_motionControl(void)
```

Die Funktion `task_motionControl` muss man ständig aus dem Hauptprogramm aufrufen – sonst funktioniert diese automatische Regelung nicht! Ständig aus dem Hauptprogramm aufrufen heisst hier nichts anderes, als dass man diese Funktion in der Hauptschleife des Programms bei jedem Schleifendurchlauf einmal aufrufen muss. Es reicht, wenn die Funktion alle 10 bis 50 Millisekunden einmal aufgerufen wird. Besser ist es allerdings, die Funktion noch sehr viel öfter aufzurufen. Es schadet nicht, wenn man die Funktion noch schneller aufruft, denn das Timing wird von einem der Hardwaretimer gesteuert. Daher ist es auch egal ob man die Funktion in gleichbleibenden Intervallen aufruft, oder ob man mal 1ms und mal 10ms zwischen den einzelnen Aufrufen benötigt. Es kostet auch nur unwesentlich mehr Rechenzeit, die Funktion oft aufzurufen. Sie wird nur komplett ausgeführt, wenn auch etwas zu tun ist.

Wird diese Funktion korrekt angewandt, kümmert sie sich darum die Motoren immer relativ genau mit den gewünschten Drehzahlen laufen zu lassen.

Das wird hier erreicht, indem bei jeder Geschwindigkeitsmessung die Abweichung ermittelt und alle gemessenen Abweichungen aufsummiert werden (also ein integrierender Regler). Mit diesem Fehlerwert kann dann die Spannung an den Motoren über die PWM Module des Mikrocontrollers angepasst werden. Ist die Geschwindigkeit zu niedrig, sind die Fehlerwerte positiv und die Spannung an den Motoren wird dementsprechend erhöht. Ist die Geschwindigkeit zu hoch, wird die Spannung verringert. Das pendelt sich beim RP6 dann schnell auf einen relativ konstanten PWM Wert ein (kleine Schwankungen sind normal). Durch diese Regelung wird die Geschwindigkeit unabhängig von der Akkuspannung, der Belastung (Gewicht, Untergrund, Steigung/Abhang etc.) und Fertigungstoleranzen eingeregelt. Würde man die PWM Kanäle nur auf einen festen Wert einstellen, würde sich die Geschwindigkeit abhängig von der Belastung und Akkusspannung sehr stark verändern und wäre auch für jeden Kanal unterschiedlich aufgrund der immer vorhandenen Fertigungstoleranzen.

Auch die Drehrichtung der Motoren wird von dieser Funktion gesteuert, denn es wäre der Lebensdauer der Motoren und der Getriebe nicht gerade zuträglich, wenn man öfters z.B. bei 15cm/s die Drehrichtung abrupt ändert. Wenn die Drehrichtung also geändert werden soll, wird automatisch auf 0cm/s abgebremst, die Drehrichtung geändert und erst dann wieder auf die Sollgeschwindigkeit beschleunigt.

Neben der Geschwindigkeitsregelung und Drehrichtungskontrolle, wird auch der Motorstrom überwacht und die Motoren automatisch abgeschaltet wenn einer von beiden zuviel Strom benötigen sollte. Das verhindert, dass die Motoren überlastet werden und

heißlaufen, was auf Dauer zu Beschädigungen führen würde. Nach drei Überstromereignissen innerhalb von 20 Sekunden wird der Roboter komplett gestoppt und die vier roten Status LEDs fangen an zu blinken.

Ebenfalls wird überwacht ob einer der Encoder oder Motoren ausgefallen ist (kann schonmal passieren, wenn man zuviel dran rumbastelt...). Denn falls das der Fall ist, würde die motionControl Funktion den PWM Wert bis aufs Maximum erhöhen und so evtl. den Roboter ausser Kontrolle geraten lassen... und das wäre natürlich weniger schön! Auch in diesem Fall wird der Roboter komplett gestoppt.

Um es übersichtlich zu halten, wurden auch die Routinen für das Abfahren von Wegstrecken und das Drehen um bestimmte Winkel mit in diese Funktion aufgenommen.

Sie sehen also, dass diese Funktion extrem wichtig für den automatischen Betrieb des Antriebssystems ist. Der motionControl Funktion selbst, werden übrigens keine Parameter wie z.B. die Sollgeschwindigkeit übergeben. Das passiert über andere Funktionen, die wir nun im einzelnen erläutern werden.

void moveAtSpeed(uint8_t desired_speed_left, uint8_t desired_speed_right)

Mit dieser Funktion wird die Sollgeschwindigkeit eingestellt. Die beiden Parameter geben die Geschwindigkeit für den linken und für den rechten Motor an. Sofern man wie oben beschrieben die motionControl Funktion ständig aufruft, wird die Geschwindigkeit auf diesen Wert eingeregelt. Setzt man den Wert auf 0, werden die PWM Module komplett deaktiviert nachdem der Roboter abgebremst hat.

getDesSpeedLeft() und **getDesSpeedRight()**

Mit diesen Makros kann man die derzeit eingestellte Sollgeschwindigkeit auslesen.

Die Anwendung der moveAtSpeed Funktion ist recht einfach – ein Beispiel:

```
#include "RP6RobotBaseLib.h"

int main(void)
{
	initRobotBase(); // Mikrocontroller initialisieren

	powerON();  // Encoder und Motorstromsensoren anschalten (WICHTIG!!!)

	moveAtSpeed(70,70); // Geschwindigkeit einstellen

	while(true)
	{
		// Aus der Hauptschleife ständig die motionControl Funktion
		// aufrufen - sie regelt die Motorgeschwindigkeiten:
		task_motionControl();
		task_ADC(); // Wird wegen den Motorstromsensoren aufgerufen!
	}
	return 0;
}
```

... und schon fährt der RP6 los! Aber er reagiert natürlich überhaupt nicht auf Hindernisse und fährt einfach nur geradeaus! Einzig die Geschwindigkeit wird möglichst konstant gehalten.

Der Roboter erhöht oder verringert ggf. auch automatisch die Motorleistung – z.B. wenn er eine Rampe hochfährt.

VORSICHT: Dieses Verhalten kann für die eigenen Finger sehr gefährlich werden – Achten Sie darauf, nicht mit den Fingern zwischen Raupenkette und Räder, oder zwischen Platine und Raupenkette zu geraten! Es besteht sonst **Verletzungsgefahr!** Wie gesagt, bei Hindernissen wird die Motorleistung automatisch erhöht und der Antrieb kann dann ziemlich viel Kraft aufbringen!

Die Geschwindigkeit wird bei der moveAtSpeed Funktion nicht in cm/s o.ä. angegeben – es wird stattdessen die Drehzahl verwendet. Das ist sinnvoller, denn die tatsächliche Geschwindigkeit ist abhängig vom Umfang der Räder/Ketten und damit von der Encoder Auflösung. Wie wir aus Kapitel 2 ja schon wissen, gibt es da einen recht großen Spielraum von etwa 0.23 bis 0.25mm pro Zählschritt und das muss man erst kalibrieren.

Die Drehzahl wird alle 200ms, also 5x pro Sekunde gemessen. Die Einheit ist daher „Encoder Segmente pro 200ms" oder anders ausgedrückt „Encoder Segmente die in 0.2s gezählt wurden". Bei einem Wert von 70, wie im obigen Beispielprogramm, wären das also etwa $70 \cdot 5 = 350$ Segmente/Zählschritte pro Sekunde (das sind etwa 8 bis 8.7 cm/s – je nach tatsächlicher Encoder Auflösung). Die minimal einregelbare Drehzahl ist etwa $10 \cdot 5 = 50$ und die maximale Drehzahl $200 \cdot 5 = 1000$. Das wird von den Motorfunktionen auf diesen Maximalwert begrenzt, wie in Kapitel 2 schon beschrieben. Wir empfehlen für den Dauerbetrieb nicht mehr als 160 einzustellen!

```
getLeftSpeed()  und  getRightSpeed()
```

Die mit den Drehgebern gemessene Drehzahl kann man über diese beiden Makros auslesen. Hier wird natürlich genau dieselbe Einheit wie oben beschrieben verwendet!

```
void changeDirection(uint8_t dir)
```

Die Bewegungsrichtung der Motoren kann mit dieser Funktion eingestellt/geändert werden. Wie oben beschrieben wird zunächst abgebremst, dann die Drehrichtung geändert und erst dann wieder auf die zuvor eingestellte Sollgeschwindigkeit beschleunigt.

Die möglichen Parameter sind:

```
FWD - für Vorwärts (FWD ist eine Abkürzung für „Forwards")
BWD - für Rückwärts (BWD ist eine Abkürzung für „Backwards")
LEFT - nach links drehen
RIGHT - nach rechts drehen
```

Über das Makro:

```
getDirection()
```

kann man die aktuell eingestelle Richtung auslesen.

Beispiel:

```
1  #include "RP6RobotBaseLib.h"
2
3  int main(void)
4  {
5          initRobotBase(); // Mikrocontroller initialisieren
6          powerON();       // Encoder und Motorstromsensoren anschalten!
7
8          moveAtSpeed(60,60); // Geschwindigkeit einstellen
9          startStopwatch1();  // Stopwatch1 starten
10
11         while(true)
12         {
13                 if(getStopwatch1() > 4000) // Sind 4000ms (= 4s) vergangen?
14                 {
15                         // Bewegungsrichtung ändern:
16                         if(getDirection() == FWD) // Fahren wir vorwärts...
17                                 changeDirection(BWD); // dann Rückwärts einstellen!
18                         else if(getDirection() == BWD) //... oder Rückwärts?
19                                 changeDirection(FWD); // dann Vorwärts einstellen!
20                         setStopwatch1(0); // Stopwatch1 auf 0 zurücksetzen
21                 }
22                 task_motionControl(); // Automatische Antriebsregelung
23                 task_ADC(); // Wird wegen den Motorstromsensoren aufgerufen!
24         }
25         return 0;
26 }
```

In diesem Beispiel fährt der RP6 zunächst geradeaus – das ist die immer nach einem Reset eingestellte Bewegungsrichtung. Wir verwenden eine der Stopwatches um 4 Sekunden zu warten und dann die Bewegungsrichtung umzuschalten. Es wird in Zeile 16 und 18 abgefragt, in welcher Richtung wir uns derzeitig bewegen und dann die jeweils andere Richtung eingestellt. Das wiederholt sich alle 4 Sekunden und so fährt der Roboter die ganze Zeit vor und zurück.

Auch hier reagiert er natürlich noch nicht auf Hindernisse!

Mit den bisher besprochenen Funktionen ist es nicht so einfach, bestimmte Distanzen zu fahren – dafür gibt es zwei spezielle Funktionen.

```
void move(uint8_t desired_speed, uint8_t dir, uint16_t distance,
                  uint8_t blocking)
```

Mit der move Funktion, kann man gerade Strecken mit vorgegebener Distanz abfahren. Man muss Sollgeschwindigkeit, Richtung (FWD oder BWD) und die Distanz in Encoder Zählschritten angeben.

Mit dem Makro:

```
DIST_MM(DISTANCE)
```

kann man von einer Distanz in Millimetern auf Encoder Zählschritte umrechnen. Dazu sollte man natürlich die Encoder Auflösung kalibriert haben (s. Anhang). Im Beispielprogramm weiter unten sieht man, wie man das anwenden kann.

Der Roboter wird versuchen die gewünschte Strecke möglichst genau abzufahren. In der motionControl Funktion wird dazu zunächst ganz normal auf die gewünschte Geschwindigkeit beschleunigt, dann aber kurz vor dem Ziel abgebremst um nicht zu weit zu fahren. Das klappt aber nur auf ein paar Millimeter genau (etwa plusminus 5mm) – was in den meisten Fällen aber gut ausreichen sollte.

Ganz kurze Strecken kann man mit der Funktion leider nicht genau fahren, es sollten schon mindestens 5cm sein. Das kann allerdings noch stark verbessert werden!

Der letzte Parameter der Funktion, „blocking" ist eine kleine Besonderheit, die wir etwas genauer erklären müssen.

Die Funktion setzt normalerweise nur ein paar Variablen und es würde dann sofort mit dem normalen Programmablauf weitergemacht. Die Fahrt würde „im Hintergrund" automatisch von der motionControl Funktion gesteuert. Das ist sinnvoll, wenn der Roboter nebenbei noch andere Dinge tun muss, wie z.B. auf Hindernisse reagieren. Aber wenn man einfach nur ein bestimmtes Muster abfahren möchte, kann man das über den blocking Parameter beeinflussen.

Ist der Parameter „true" (also 1) ruft die Funktion in einer Schleife solange die motionControl Funktion auf, bis die gewünschte Strecke abgefahren worden ist. Die Funktion wird dann nicht einfach verlassen, sondern blockiert den normalen Programmablauf für diese Zeit.

Ist der Parameter false, verhält sich die Funktion wie oben beschrieben und wird sofort verlassen, nachdem der „Auftrag zum Fahren einer Strecke" erteilt wurde. Wenn man dann während dieser Zeit andere Funktionen aufruft, die z.B. die Geschwindigkeit setzen, neue Fahrbefehle geben o.ä., könnte das Programm nicht wie gewünscht funktionieren. Man muss entweder solange warten bis die Strecke abgefahren wurde, oder den Vorgang vorzeitig abbrechen.

Mit der Funktion

```
uint8_t isMovementComplete(void)
```

kann man überprüfen ob der Bewegungsvorgang abgeschlossen ist. Liefert die Funktion false zurück, ist die Bewegung noch im Gange, andernfalls ist sie abgeschlossen und es können neue Befehle gegeben werden.

Muss man die Bewegung vorzeitig abbrechen, weil z.B. ein Hindernis entdeckt wurde, kann man mit der Funktion:

```
void stop(void)
```

alle Bewegungsabläufe abbrechen und den Roboter anhalten.

Jetzt könnte man die oben genannte Funktion sogar schon für das Rotieren um bestimmte Winkel verwenden, indem man einfach für den Richtungsparameter LEFT oder RIGHT anstatt FWD oder BWD angibt und dann die Distanz angibt die zum gewünschten Winkel passt. Das ist aber umständlich und funktioniert außerdem nicht besonders gut. Daher gibt es eine eigene Funktion, die auf die Rotation auf der Stelle zugeschnitten ist:

```
void rotate(uint8_t desired_speed, uint8_t dir, uint16_t angle,
                 uint8_t blocking)
```

Das funktioniert genau wie die move Funktion. Einziger Unterschied ist, dass die Angabe der zu fahrenden Distanz durch einen Winkel ersetzt worden ist. Auch das Verhalten dieser Funktion wird über den Parameter blocking gesteuert.

An diesem Beispiel sieht man, wie die beiden Funktionen benutzt werden können:

```
#include "RP6RobotBaseLib.h"

int main(void)
{
	initRobotBase();
	setLEDs(0b111111);
	mSleep(1500);

	powerON(); // Encoder und Motorstromsensoren anschalten!

	while(true)
	{
		setLEDs(0b100100);
		move(60, FWD, DIST_MM(300), true); // 30cm Vorwärts fahren
		setLEDs(0b100000);
		rotate(50, LEFT, 180, true);  // um 180° nach links drehen
		setLEDs(0b100100);
		move(60, FWD, DIST_MM(300), true); // 30cm Vorwärts fahren
		setLEDs(0b000100);
		rotate(50, RIGHT, 180, true); // um 180° nach rechts drehen
	}
	return 0;
}
```

Hier fährt der Roboter zunächst 30cm Vorwärts, dreht dann um 180° nach links, fährt die 30cm zurück, dreht um 180° nach rechts und das Spiel beginnt von neuem. Würde man den blocking Parameter von allen Funktionen false anstatt true setzen, würde das Programm überhaupt nicht funktionieren, da aus der Main Funktion nicht die task_motionControl Funktion aufgerufen wird und alle Funktionsaufrufe direkt hintereinander stehen. Ändert man auch nur einen der blocking Parameter in false um, wird das Programm ebenfalls nicht mehr wie gewünscht funktionieren – es wird dann eine der Bewegungsphasen komplett übersprungen.

Für solche rein sequentiellen Abläufe muss man den blocking Parameter also immer auf true setzen!
Man kann sogar rudimentär auf Hindernisse und andere Dinge reagieren wenn der blocking Parameter auf true gesetzt ist – nämlich über die Event Handler. Diese werden trotzdem aufgerufen! Das klappt aber nicht unbedingt für komplexere Problemstellungen.

Allgemein sollte man wenn man Hindernissen ausweichen, Befehle von einem Master Controller interpretieren oder ähnliche Dinge erledigen will, den non-blocking Modus verwenden, den Parameter also auf false setzen.
Wie oben schon erwähnt, regeln die move/rotate Funktionen dann unabhängig vom restlichen Programmablauf die Fahrt des Roboters.

Auf der CD finden sich einige umfangreichere Beispiele zu diesen Funktionen.

4.6.10. task_RP6System()

Wie wir in den vorangegangenen Abschnitten gesehen haben, muss man vier Funktionen ständig aus der Hauptschleife heraus aufrufen, damit ACS/IRCOMM, Motoren und Bumper korrekt funktionieren und die ADCs im Hintergrund ausgewertet werden. Um etwas Schreibarbeit zu sparen und es übersichtlicher zu machen, wurde die Funktion:

```
void task_RP6System(void)
```

implementiert. Sie ruft die Funtionen:

```
task_ADC();
task_ACS();
task_bumpers();
task_motionControl();
```

der Reihe nach auf. In den meisten Beispielprogrammen auf der CD werden Sie nur diese Funktion sehen – die anderen vier werden nur selten direkt verwendet und meistens über diese Funktion aufgerufen.

4.6.11. I²C Bus Funktionen

Zum Schluss dieses Kapitels, kommen wir zu den I²C-Bus Funktionen. Diese sind zur Kommunikation mit anderen Mikrocontrollern und Erweiterungsmodulen notwendig.

Es gibt zwei Versionen der I²C Funktionen – eine für den Slave und eine für den Master Modus.

Achtung: *Sie können nicht beide Versionen gleichzeitig verwenden!*

Man kann jeweils eine der beiden Versionen einbinden und muss diese auch im Makefile eintragen! Dort sind die passenden Einträge schon vorhanden – *aber auskommentiert*. Es darf nur einer von beiden Einträgen verwendet werden, sonst gibt der Compiler einen Fehler aus (weil der TWI Interrupt Vektor dann doppelt definiert wäre...)!

4.6.11.1. I²C Slave

Bei der Roboterbasis ist der Slave Modus sehr wichtig, denn oft wird man wohl einen weiteren Mikrocontroller zur Steuerung des Roboters verwenden. Es gibt auch ein Beispielprogramm, das so gut wie alle Funktionen der Roboterbasis über den I²C Bus bereitstellt (RP6Base_I2CSlave).

Der Slave Modus ist genau wie der Master Modus Interrupt basiert. Beim Slave ist das nicht anders möglich (ohne viel Aufwand jedenfalls), beim Master könnte man das allerdings auch gut ohne realisieren. Um es jedoch einheitlicher zu gestalten, sind beide Versionen Interrupt basiert. Nebenbei kann man die Übertragungen im Mastermodus dann auch zum Teil im Hintergrund laufen lassen, was etwas Zeit sparen kann.

void I2CTWI_initSlave(uint8_t address)

Diese Funktion initialisiert das TWI Modul des Mikrocontrollers als I²C Slave. Man kann hier die Adresse des Controllers festlegen. Die Adresse gibt auch gleich mit an, ob der Controller auf sog. „General Calls" reagieren soll. Wird der I²C Bus mit 0 adressiert, nennt man das General Call und es können alle angeschlossenen Geräte darauf reagieren. Das ist z.B. nützlich um alle Controller gleichzeitig in einen Stromsparmodus zu schalten o.ä.

Beispiele:

```
I2CTWI_initSlave( adr |  TWI_GENERAL_CALL_ENABLE ); // General call aktiv
I2CTWI_initSlave(adr); // General call NICHT aktiv
```

I²C Register

Gängige I²C Bus Peripherie wird meist über einige Register die man lesen und/oder beschreiben kann gesteuert. Die Slave Routinen sind deshalb auch so gestaltet, dass es „Register" (in diesem Fall ein Array aus 8 Bit Variablen) gibt, die der Bus-Master beschreiben bzw. lesen kann. Der Master überträgt dazu nach der Slave Adresse noch die Registeradresse und kann dann Daten aus diesem Register lesen oder es beschreiben.

Es gibt zwei Arrays mit dem Datentyp uint8_t dafür. Eines davon stellt die lesbaren Register zur Verfügung, das andere die beschreibbaren Register.

Die beiden Arrays und die Variable für General Calls heissen

I2CTWI_readRegisters, I2CTWI_writeRegisters und I2CTWI_genCallCMD

I2CTWI_readRegisters sind die lesbaren Register und I2CTWI_writeRegisters sind die beschreibbaren Register. I2CTWI_genCallCMD speichert das zuletzt empfangene General Call Kommando. Der Datenaustausch über den I²C Bus läuft im Slave Modus komplett über diese Register. Wenn man Daten auf dem Bus verfügbar machen will, muss man diese im Array I2CTWI_readRegisters ablegen. Dann kann ein Master über die Arrayposition (=Registernummer) auf die gewünschten Daten zugreifen. Soll ein Master z.B. Sensorwerte vom Slave empfangen, schreibt man diese einfach an vorher festgelegte Positionen im I2CTWI_readRegisters Array. Anschließend kann der Master diese Daten auslesen, indem er zunächst die Registernummer an den Slave überträgt und dann die Daten liest. Die Registernummer wird automatisch inkrementiert, also kann der Master auch mehrere Register in einem Rutsch auslesen.

Ähnlich funktioniert auch das Schreiben. Der Master überträgt zunächst die gewünschte Zielregisternummer und fängt dann an, die Daten zu schreiben. Wenn man mehrere Register hintereinander beschreiben will, muss man auch hier nicht für jedes Register einzeln die Nummer übertragen, sondern kann einfach weiterschreiben da die Registernummer nach jedem Zugriff automatisch inkrementiert wird.

Das passiert auf dem Slave komplett Interrupt basiert im Hintergrund.

Beim Schreiben auf den Slave kann es leicht zu Inkonsistenzen kommen, wenn man die Daten direkt aus den Registern benutzt. Denn wenn man ein Register ausliest, könnte der Master ein anderes bereits überschrieben haben. Oft ist es daher sinnvoll, die Daten vor der Verwendung zwischenzuspeichern. Auch Beim Lesen kann es zu inkonsistenzen kommen, wenn mehrere Variablen voneinander abhängen (z.B. low und high Byte einer 16 Bit Variablen).

I2CTWI_readBusy und I2CTWI_writeBusy

Die Interrupt Routine setzt bei Schreibzugriffen des Masters die Variable I2CTWI_writeBusy auf true – so kann man überprüfen, ob gerade keine Schreibzugriffe stattfinden und kann danach die Daten aus den Registern in temporäre Variablen schreiben, mit denen man weiterarbeiten kann.

Im Beispielprogramm auf der nächsten Seite und im Slave Beispielprogramm auf der CD wird das auch demonstriert – hier gibt es ein Kommandoregister, über das der Master dem Slave Befehle wie „fahre mit Geschwindigkeitswert 100 Vorwärts“ senden kann. Dazu wird in der Hauptschleife vom Slave ständig das Register 0 ausgewertet wenn I2CTWI_busy false ist. Ist in Register 0 ein Kommando vom Master eingetroffen, werden Register 0 und auch die Register 1 bis 6 in temporäre Variablen geschrieben. Diese können dann ausgewertet werden. Je nach Inhalt der Kommandovariablen, werden dann die Parameter behandelt. Parameter 1 könnte z.B. bei einem Bewegungsbefehl die Geschwindigkeit enthalten und Parameter 2 die Bewegungsrichtung. Die anderen Parameter würden bei diesem Befehl dann nicht weiter beachtet.

Genauso funktioniert I2CTWI_readBusy – diese Variable wird gesetzt, wenn gerade Register gelesen werden. Damit kann man also überprüfen, ob man die Register beschreiben kann, ohne Inkonsistenzen hervorzurufen. In der aktuellen Implementierung ist das aber nicht zu 100% garantiert. Um Inkonsistenzen komplett zu vermeiden, müsste man für die Zeit in der die Register beschrieben werden, den TWI Interrupt deaktivieren. Das könnte allerdings wiederum andere Probleme verursachen...

Hier ein kleines Beispielprogramm dazu:

```
1  #include "RP6RobotBaseLib.h"
2  #include "RP6I2CslaveTWI.h"  // I2C Library Datei einbinden (!!!)
3     // ACHTUNG: Das muss auch im Makefile eingetragen werden (!!!)
4  
5  #define CMD_SET_LEDS 3 // LED Kommando, das über den I2C Bus
6                         // empfangen werden soll.
7  int main(void)
8  {
9        initRobotBase();
10       I2CTWI_initSlave(10); // TWI initialisieren und Adresse 10 einstellen
11       powerON();
12 
13       while(true)
14       {
15             // irgendein Kommando empfangen und KEIN Schreibzugriff aktiv?
16             if(I2CTWI_writeRegisters[0] && !I2CTWI_writeBusy)
17             {
18                   // Register speichern:
19                   uint8_t cmd = I2CTWI_writeRegisters[0];
20                   I2CTWI_writeRegisters[0] = 0; // und zurücksetzen (!!!)
21                   uint8_t param = I2CTWI_writeRegisters[1]; // Parameter
22 
23                   if(cmd == CMD_SET_LEDS) // LED Kommando empfangen?
24                         setLEDs(param); // LEDs mit dem Parameter setzen
25             }
26             if(!I2CTWI_readBusy) // Kein Lesezugriff aktiv?
27                   // Dann aktuellen LED Zustand ins Register 0 schreiben:
28                   I2CTWI_readRegisters[0] = statusLEDs.byte;
29       }
30       return 0;
31 }
```

Das Programm tut von sich aus erstmal nichts. Man braucht also auf jeden Fall einen Master der den Controller als Slave steuert. Der Slave ist hier unter Adresse 10 auf dem I²C Bus erreichbar (s. Zeile 10). Es gibt nur zwei Register die man beschreiben kann und ein Register das man lesen kann. Das erste Register (= Registernummer 0) ist ein Kommandoregister in dem Befehle empfangen werden können. In diesem simplen Beispiel ist dies nur das Kommando „3“, um die LEDs zu setzen (kann irgendeine beliebige Nummer sein). Wird ein beliebiges Kommando empfangen und ist kein Schreibzugriff aktiv (Zeile 16), wird zunächst der Wert des Kommandoregisters in der Variablen cmd gespeichert (Zeile 19) und danach das Kommandoregister zurückgesetzt, damit das Kommando nicht erneut ausgeführt wird! Dann wird noch der Parameter in Register 1 in einer Variablen gesichert und abgefragt, ob das Kommando 3 empfangen wurde (Zeile 23). Ist das der Fall, werden die LEDs mit dem Wert vom Parameter gesetzt.
In Zeile 26 wird abgefragt ob gerade kein Schreibzugriff stattfindet und dann ggf. der aktuelle Wert der LEDs im lesbaren Register 0 gespeichert.
Wenn der Controller auf dem Roboter dieses Programm geladen hat, kann ein Mastercontroller also die LEDs auf dem Roboter über den I²C Bus setzen und deren aktuellen Zustand auslesen.

Sonst tut das Programm nichts – ein ausführlicheres Beispiel bei dem man so gut wie alles auf dem Roboter steuern kann gibt es wie schon gesagt auf der CD. Das hier demonstriert nur das Funktionsprinzip.

Standardmäßig gibt es 16 beschreibbare und 48 lesbare Register. Wer mehr oder weniger benötigt, kann diese Werte in der Datei RP6I2CSlaveTWI.h der RP6Library anpassen.

4.6.11.2. I²C Master

Im Master Modus kann man mit dem TWI Modul des MEGA32 andere Geräte/Mikrocontroller/Sensoren über den I²C Bus steuern.

```
void I2CTWI_initMaster(FREQ)
```

Diese Funktion initialisiert das TWI Modul als Master. Für den Master Modus braucht man natürlich keine Adresse anzugeben – aber man muss eine Frequenz festlegen, mit der das TWI Modul die Daten senden soll. Über den Parameter FREQ kann die Frequenz in kHz eingestellt werden. Meistens verwendet man 100kHz, also tragen Sie hier am besten immer eine 100 ein. Wenn es etwas schneller gehen soll, kann man auch bis zu 400 eintragen. Mehr als 400 sollten Sie hier nicht eintragen – das ist die maximal spezifizierte Geschwindigkeit des TWI Moduls.

Das TWI Modul des MEGA32 auf dem Mainboard kann allerdings im Master Modus nach Atmel Spezifikation nur bis etwa 220kBit/s verwendet werden (s. Datenblatt)! Der Controller wird aus Stromspargründen mit 8MHz getaktet, für 400kBit/s wären aber mehr als 14.4MHz Takt nötig. Bei unter 14.4MHz kann es ein leicht von den Spezifikationen abweichendes Timing geben. Was für die meisten I²C Geräte nicht besonders wichtig ist. Für den Slave Modus spielt das ohnehin keine Rolle, hier sind 400kBit/s problemlos möglich! Für alle normalen Anwendungen sind die garantiert möglichen 220kBit/s auch mehr als schnell genug. Wenn höhere Geschwindigkeiten wirklich dringend benötigt werden, kann das RP6 CONTROL M32 Erweiterungsmodul verwendet werden – es kann die maximalen 400kBit/s im Master Modus nutzen. Oder Sie probieren einfach, ob die Geräte die auf Ihrem Roboter am Bus angeschlossen sind, mit dem vom MEGA32 erzeugten Timing problemlos klar kommen.

Senden von Daten

Es gibt mehrere Funktionen um Daten über den I²C Bus zu übertragen. Im Prinzip funktionieren alle sehr ähnlich, sie unterscheiden sich nur in der Anzahl der zu übertragenen Bytes.

```
void I2CTWI_transmitByte(uint8_t adr, uint8_t data)
```

Überträgt ein einzelnes Byte an die angegebene Adresse.

```
void I2CTWI_transmit2Bytes(uint8_t adr, uint8_t data1, uint8_t data2)
```

Überträgt zwei Bytes and die angegebene Adresse.

Diese Funktion braucht man häufig, denn viele I²C Geräte erwarten die Daten im Format:

Slave Adresse – Registeradresse – Daten

```
void I2CTWI_transmit3Bytes(uint8_t adr, uint8_t data1, uint8_t data2,
uint8_t data3)
```

Braucht man ebenfalls häufig. Vor allem im Zusammenhang mit dem oben beschriebenen Slave Programm – hier kann man die Daten im Format

Slave Adresse – Kommandoregister – Kommando – Parameter1

übertragen.

```
void I2CTWI_transmitBytes(uint8_t targetAdr, uint8_t *msg,
                          uint8_t numberOfBytes)
```

Diese Funktion überträgt standardmäßig bis zu 20 Bytes an die angegebene Adresse. Wer mehr Bytes direkt hintereinander übertragen muss, kann in der Header Datei den Eintrag I2CTWI_BUFFER_SIZE vergrößern.

Falls man ein Register angeben muss, an das die Daten geschickt werden, kann man einfach das erste Byte des Buffers verwenden.

Die Anwendung dieser Funktionen ist denkbar einfach. Beispiel:

```
I2CTWI_transmit2Bytes(10, 2, 128);
I2CTWI_transmit2Bytes(10, 3, 14);
I2CTWI_transmit3Bytes(64, 12, 98, 120);
```

Hier werden zweimal nacheinander zwei Bytes an den Slave mit der Adresse 10 geschickt und dann noch drei an den Slave mit Adresse 64.

Analog dazu funktionieren die anderen transmitXBytes Funktionen.

Weiteres Beispiel:

```
uint8_t messageBuf[4];
messageBuf[0] = 2; // Evtl. hier das zu beschreibende Register angeben...
messageBuf[1] = 244;  // Daten...
messageBuf[2] = 231;
messageBuf[3] = 123;
messageBuf[4] = 40;
I2CTWI_transmitBytes(10,&messageBuf[0],5);
```

So kann man mehrere Bytes (hier 5) nacheinander über den I²C Bus schicken.

Die genannten Funktionen blockieren den Programmablauf nur dann, wenn das I²C Interface gerade aktiv ist. Dann wird solange gewartet bis die Übertragung komplett ist. Wenn man das abfragt bevor man die Funktion aufruft, kann man so Daten übertragen und während der Übertragung andere Dinge tun.

Die Transfers über den I²C Bus dauern schließlich relativ lange verglichen mit der Geschwindigkeit des Mikrocontrollers.

Das Makro

```
I2CTWI_isBusy()
```

gibt zurück, ob das TWI Modul derzeit verwendet wird. Falls nicht, kann man neue Daten übertragen.

Empfangen von Daten

Zum Empfangen von Daten bietet die RP6Library mehrere Möglichkeiten. Zum einen blockierende Funktionen, die ähnlich wie die Schreibfunktionen aufgebaut sind. Zum anderen aber auch Funktionen, die die Daten automatisch „im Hintergrund" abholen.

Zunächst zu der einfachen Variante, Daten blockierend lesen.

```
uint8_t I2CTWI_readByte(uint8_t targetAdr);
```

Diese Funktion liest ein einzelnes Byte von einem Slave. Einzeln kann man die Funktion nicht verwenden, man muss fast immer erst noch die Registernummer übertragen. Das macht man mit I2CTWI_transmitByte.

Beispielsweise wenn man Register 22 vom Slave mit Adresse 10 lesen will:

```
I2CTWI_transmitByte(10, 22);
uint8_t result = I2CTWI_readByte(10);
```

```
void I2CTWI_readBytes(uint8_t targetAdr, uint8_t * messageBuffer,
                      uint8_t numberOfBytes);
```

Will man mehrere Bytes abrufen, kann man diese Funktion verwenden.

Beispiel:

```
I2CTWI_transmitByte(10, 22);
uint8_t results[6];
I2CTWI_readBytes(10,results,5);
```

Hier werden 5 Bytes von Register 22 des Slaves mit der Adresse 10 gelesen. Ob diese wirklich von Register 22 gelesen werden, variiert von Slave zu Slave – einige inkrementieren automatisch die Registernummer (wie der Slave Code der RP6Library) und andere Funktionieren ganz anders. Dazu müssen Sie immer die jeweilige Dokumentation lesen!

Will man Daten im Hintergrund auslesen, gestaltet sich die Sache schon ein wenig komplexer. Man startet zunächst eine Anfrage (Request) nach einer bestimmten Anzahl von Bytes von einem bestimmten Slave. Diese Bytes werden dann im Hintergrund vom Slave abgeholt. Der Controller kann in dieser Zeit also etwas anderes tun und wird nur ab und zu kurz von der Interruptroutine unterbrochen. Man muss allerdings ständig aus der Hauptschleife eine Funktion aufrufen, die überprüft ob die angeforderten Daten vom Slave empfangen worden sind, oder evtl. ein Fehler aufgetreten ist. Ist das der Fall, ruft diese Funktion automatisch zuvor eingestellte Event Handler Funktionen auf und man kann die empfangenen Daten weiterverarbeiten und ggf. sofort weitere Daten aus anderen Registern anfordern. Jeder Request bekommt dabei eine ID, damit man diese auch zuordnen kann.

```
void task_I2CTWI(void)
```

Dies ist die Funktion die ständig aus der Hauptschleife aufgerufen werden muss. Sie überprüft ob der Transfer komplett fehlerfrei abgeschlossen wurde und ruft dementsprechend die Event Handler auf.

```
void I2CTWI_requestDataFromDevice(uint8_t requestAdr, uint8_t requestID,
                                  uint8_t numberOfBytes)
```

Und mit dieser Funktion kann man Daten von einem Slave anfordern. Nachdem man diese Funktion aufgerufen hat, läuft der Abholvorgang wie oben beschrieben automatisch im Hintergrund.

Anschließend kann man die Daten mit der Funktion:

```
void I2CTWI_getReceivedData(uint8_t *msg, uint8_t msgSize)
```

abrufen.

Man kann die Daten abrufen, sobald der passende Event Handler aufgerufen wird. Diesen registriert man mit der Funktion:

```
void I2CTWI_setRequestedDataReadyHandler(void (*requestedDataReadyHandler)
(uint8_t))
```

Der Event Handler muss die Signatur

```
void I2C_requestedDataReady(uint8_t dataRequestID)
```

haben. Dem Event Handler wird die ID übergeben, die man der jeweiligen Datenanforderung zugewiesen hat. Darüber kann man Zugriffe auf verschiedene Slaves unterscheiden.

Neben dem requestedDataReady Handler, gibt es auch noch einen Event Handler, der bei Fehlern aufgerufen wird. Wenn während der Übertragung irgendetwas schief geht, z.B. wenn der Slave nicht antwortet, wird dieser Event Handler aufgerufen.

Man registriert den Event Handler mit dieser Funktion:

```
void I2CTWI_setTransmissionErrorHandler(void (*transmissionErrorHandler)
(uint8_t))
```

Er muss die Signatur:

```
void I2C_transmissionError(uint8_t errorState)
```

haben. Der Parameter ist ein Fehlerzustandscode. Die Bedeutung der Fehlercodes kann man der Header Datei des I²C Master Modus entnehmen.

Diesen Event Handler kann man übrigens unabhängig davon verwenden, ob man nun die Daten im Hintergrund abruft oder nicht. Auch mit den blockierenden Funktionen wird dieser Event Handler bei einem Fehler aufgerufen.

Einige Beispielprogramme zum Master Modus finden Sie auf der CD – diese werden auch im folgenden Kapitel über die Beispielprogramme noch genauer besprochen.

4.7. Beispielprogramme

Auf der CD finden Sie einige kleine Beispielprogramme. Diese Beispielprogramme demonstrieren die grundlegenden Funktionen des Roboters. Die meisten davon sind weder besonders komplex, noch stellen sie die optimale Lösung dar und verstehen sich nur als Ausgangspunkt für eigene Programme. Das ist absichtlich so, damit Ihnen auch noch etwas zu tun bleibt – wäre ja langweilig einfach nur vorgefertige Programme auszuprobieren...

Ein paar der Beispielprogramme richten sich dann aber doch eher an fortgeschrittene Anwender. Das trifft vor allem auf die Programme für verhaltensbasierte Roboter zu, wo sich der RP6 wie ein Insekt verhalten soll. In einem Beispielprogramm verhält er sich z.B. *ähnlich* wie eine Motte, die helles Licht sucht und Hindernissen ausweicht. Das alles noch genauer zu erläutern, sprengt den Rahmen dieser Bedienungsanleitung bei weitem und spätestens hier müssen wir auf die Literatur verweisen.

Sie können Ihre eigenen Programme selbstverständlich mit anderen Anwendern über das Internet austauschen. Die RP6Library und alle Beispielprogramme stehen unter der Open Source Lizenz „GPL" (General Public License) und daher sind Sie berechtigt, die Programme unter den Bedingungen der GPL zu modifizieren, zu veröffentlichen und anderen Anwendern zur Verfügung zu stellen, sofern Sie Ihre Programme dann ebenfalls unter die GPL Lizenz stellen.

Allgemein gibt es für den MEGA32 schon sehr viele Bespielprogramme im Internet, da die Controller aus der AVR Controllerfamilie bei Hobby Anwendern sehr beliebt sind. Allerdings muss man hier immer darauf achten, andere Beispielprogramme auch an die Hardware des RP6 und die RP6Library anzupassen – sonst wird es oft nicht funktionieren (die offensichtlichsten Probleme sind andere Pinbelegungen, Verwendung von bereits anderweitig verwendeten Hardwaremodulen wie Timern, andere Taktfrequenz, etc. pp.)!

Alle Beispielprogramme bis auf diejenigen die den I²C Bus verwenden, sind darauf ausgelegt nur auf der Roboterbasis zu laufen – also ohne Erweiterungsmodule. Auch wenn das normalerweise nicht weiter stört, sollten Sie Erweiterungsmodule erst dann auf dem Roboter anbringen, wenn Sie alle Beispielprogramme mal ausprobiert und sich mit den Möglichkeiten der Roboterbasis vertraut gemacht haben.

Bei jedem programmierbaren Erweiterungsmodul sollten entsprechende Beispielprogramme mitgeliefert werden, oder aber im Internet auf unserer Homepage verfügbar sein (für das RP6 CONTROL M32 sind sie sogar schon auf der RP6 CD enthalten!).

Übrigens ist die Programmierung vieler Erweiterungsmodule etwas einfacher, weil es hier oft nur wenige besonders zeitkritische Dinge gibt, die man quasi simultan ausführen muss. Das macht die grundlegende Programmierung z.B. des RP6-M32 einfacher und man muss nicht ganz so viele Besonderheiten beachten.

Nebenbei ist auf dem RP6-M32 auch viel mehr Rechenzeit und Speicher für andere Dinge frei.

Beispiel 1: „Hello World"-Programm mit LED Lauflicht
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_01_LEDs
Datei: RP6Base_LEDs.c

Das Programm erzeugt Ausgaben auf der seriellen Schnittstelle, sie sollten den Roboter also an den PC anschließen und sich die Ausgaben im Terminal der RP6Loader Software ansehen!

Der Roboter bewegt sich in diesem Beispielprogramm *nicht*! Sie können ihn also z.B. auf einen Tisch neben dem Computer stellen.

Dieses Beispielprogramm gibt einen kurzen „Hello World" Text über die serielle Schnittstelle aus und anschließend wird ein Lauflicht ausgeführt.

Eine kleine Besonderheit bei dem Lauflicht ist der „shift left" Operator, der auch schon weiter oben vorgekommen ist, ohne das wir diesen genauer erklärt haben:

```
1  setLEDs(runningLight); // LEDs setzen
2  runningLight <<= 1;    // Nächste LED (shift-left Operation)
3  if(runningLight > 32)  // Letzte LED?
4      runningLight = 1;  // Ja, also wieder von vorn beginnen!
```

Das holen wir jetzt noch kurz nach. Mit einer shift left Operation, also „<<" (Zeile 2) kann man die Bits in einer Variablen um eine gegebene Anzahl von Stellen nach links verschieben. Das geht natürlich auch nach rechts – mit dem shift right Operator „>>".

Also verschiebt `runningLight <<= 1;` alle Bits in der Variable runningLight um eine Position nach links. Die obige Zeile ist übrigens nur eine Abkürzung für

```
runningLight = runningLight << 1;
```

genau wie man es z.B. mit += oder *= tun kann.

Die Variable runningLight hat zunächst den Wert 1, also ist nur Bit 1 in der Variablen gesetzt. Mit jeder shift Operation wandert dieses Bit um eine Position nach links. Da wir diese Variable verwenden um die LEDs zu setzen, resultiert dies in einem „Laufenden" Lichtpunkt --> ein Lauflicht! Damit das menschliche Auge diesen Lichtpunkt auch wahrnehmen kann, wird mSleep verwendet um 100ms Pause nach jedem Schleifendurchlauf zu erzeugen.

Ist Bit 7 in der Variablen runningLight gesetzt (also hat sie einen Wert > 32), müssen wir wieder an den Anfang zurückspringen und nur Bit 1 setzen (Zeile 3+4).

Beispiel 2: Mehr von der seriellen Schnittstelle
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_02_UART_01
Datei: RP6Base_SerialInterface_01.c

Das Programm erzeugt Ausgaben auf der seriellen Schnittstelle

Der Roboter bewegt sich in diesem Beispielprogramm *nicht*!

Hier demonstrieren wir die Verwendung der writeInteger und writeIntegerLength Funktionen und geben ein paar Zahlenwerte in verschiedenen Formaten über die serielle Schnittstelle aus.

Ausserdem wird hier die in Version 1.3 der RP6Lib neu eingeführte variable „timer" demonstriert, die für Zeitmessungen mit einer Auflösung von 100µs verwendet werden kann.

Beispiel 3: Frage-Antwort Programm
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_02_UART_02
Datei: RP6Base_SerialInterface_02.c

Das Programm erzeugt Ausgaben auf der seriellen Schnittstelle

Der Roboter bewegt sich in diesem Beispielprogramm *nicht*!

Dieses etwas umfangreichere Beispiel ist ein kleines Frage-Antwort Programm. Der Roboter stellt vier einfache Fragen und man kann diese beliebig beantworten. Der Roboter reagiert z.B. mit einer Textantwort oder er führt eine kleine Lauflichtroutine aus. Das Programm soll eigentlich nur demonstrieren wie man Daten von der seriellen Schnittstelle empfangen und weiterverarbeiten kann. Nebenbei wird auch die writeStringLength Funktion demonstriert und es ist ein weiteres Beispiel zum switch-case Konstrukt.

Die alten Funktionen zum Datenempfang über die serielle Schnittstelle wurden in einer neueren Version der RP6Lib durch deutlich bessere ersetzt. Dieses Beispielprogramm demonstriert direkt deren Anwendung. Jetzt ist es relativ einfach möglich, komplette Eingabezeilen zu verarbeiten und besser auf fehlerhafte Eingaben zu reagieren.

Beispiel 4: Stopwatches Demo Programm
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_03_Stopwatches
Datei: RP6Base_Stopwatches.c

Das Programm erzeugt Ausgaben auf der seriellen Schnittstelle

Der Roboter bewegt sich in diesem Beispielprogramm *nicht*!

In diesem Programm werden vier Stopwatches verwendet. Eine um ein Lauflicht mit 100ms Intervall zu erzeugen und die anderen für drei verschiedene Counter die Ihren Zählstand in unterschiedlichen Intervallen erhöhen und dann über die serielle Schnittstelle ausgeben.

Beispiel 5: ACS & Bumper Demo Programm
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_04_ACS
Datei: RP6Base_ACS.c

Das Programm erzeugt Ausgaben auf der seriellen Schnittstelle

Der Roboter bewegt sich in diesem Beispielprogramm *nicht*!

Auch wenn der Dateiname nur auf das ACS hindeutet, dieses Programm demonstriert die Verwendung von ACS *und* Bumpern mit passenden Event Handlern. Es stellt den Zustand der beiden ACS Kanäle mit den LEDs dar und gibt diesen auch auf der seriellen Schnittstelle aus. Der Zustand der Bumper wird nur auf der seriellen Schnittstelle ausgegeben.

Bewegen Sie einfach mal die Hand vor dem Roboter und drücken sie die Bumper!

Sie können das Programm verwenden um verschiedene Gegenstände mit dem ACS zu testen – also welche gut und welche eher schlecht erkannt werden. Sie können auch die Sendeleistung des ACS verändern! Standardmäßig ist es auf die mittlere Stufe eingestellt.

Beispiel 6: Der Roboter fährt im Kreis
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_05_Move_01
Datei: RP6Base_Move_01.c

ACHTUNG: Der Roboter bewegt sich in diesem Beispielprogramm! Sie müssen also die Verbindung zum PC nach dem Programmupload trennen und den Roboter auf eine ausreichend große Fläche Stellen! Es sollte schon mindestens eine Fläche von 1m x 1m oder eher 2m x 2m sein.

Jetzt kann der Roboter endlich mal seine Motoren anwerfen! Dieses Beispielprogramm lässt den Roboter im Kreis fahren, indem die beiden Motoren auf unterschiedliche Geschwindigkeiten eingestellt werden. Stellen Sie bitte für alle Programme in denen sich der Roboter bewegt genügend freie Fläche zur Verfügung! Ein oder zwei Quadratmeter sollten es bei einigen Programmen schon mindestens sein.

Sollte der Roboter während seiner Kreisfahrt mit den Bumpern gegen ein Hindernis fahren, werden die Motoren gestoppt und zwei der LEDs fangen an zu blinken! Der Roboter tut dann nichts anderes mehr, bis das Programm neu gestartet wird.

Beispiel 7: Der Roboter fährt hin und her - mit 180° Drehung
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_05_Move_02
Datei: RP6Base_Move_02.c

ACHTUNG: Der Roboter bewegt sich in diesem Beispielprogramm!

Im RP6Library Kapitel hatten wir ja schon ein kleines Beispiel bei dem der Roboter Zeitgesteuert hin und hergefahren ist. In diesem Beispiel, fährt der Roboter eine bestimmte Distanz vorwärts, dreht um 180°, fährt diese Distanz zurück, dreht wieder um 180° und das ganze beginnt wieder von vorn.

Hier wird der blockierende Modus der move und rotate Funktionen verwendet, daher braucht man also die Funktion task_RP6System hier nicht ständig aufzurufen, weil das schon automatisch von diesen Funktionen gemacht wird.

Beispiel 8: Der Roboter fährt im Quadrat
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_05_Move_03
Datei: RP6Base_Move_03.c

ACHTUNG: Der Roboter bewegt sich in diesem Beispielprogramm!

Nun versucht der Roboter, Quadrate mit 30cm Kantenlänge abzufahren. Er dreht sich nach jedem kompletten Umlauf um und fährt in die andere Richtung weiter.

Das klappt nur mit genau kalibrierter Drehgeberauflösung ausreichend gut, weil sonst die 90° Winkel nicht genau getroffen werden und evtl. auch z.B. mal 98° oder 80° sein können. An diesem Beispiel sieht man selbst mit gut kalibrierten Drehgebern, dass es nicht einfach ist die Fahrt eines Roboters mit Raupenantrieb anhand der Drehgebermesswerte zu steuern – wie in Kapitel 2 ja schon gesagt wurde. Die Ketten rutschen je nach Untergrund vor allem bei Drehungen gern mal etwas zur Seite weg oder drehen durch, so dass die tatsächlich gefahrene Wegstrecke deutlich kürzer ist, als die von den Encodern gemessene Wegstrecke. Das kann man mit geschickter Programmierung zwar halbwegs ausgleichen, aber 100%ig genau wird das vermutlich nie werden. Das kann man nur mit zusätzlichen Sensoren erreichen, die die Position und Ausrichtung des Roboters genauer bestimmen (s. Anhang).

Beispiel 9: Exkursion - Endliche Automaten
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_05_Move_04_FSM
Datei: RP6Base_Move_04_FSM.c

Das Programm erzeugt Ausgaben auf der seriellen Schnittstelle

Der Roboter bewegt sich in diesem Beispielprogramm *nicht*!

Für komplexere Programme reichen die einfachen Methoden die wir in den bisherigen Beispielprogrammen verwendet haben nicht immer aus.

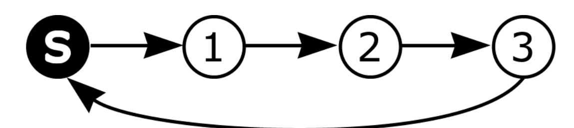
Beispielsweise für verhaltensgesteuerte Roboter braucht man endliche Automaten (engl. Finite State Machines – oder kurz FSMs) damit man das überhaupt realisieren kann. Dieses Beispielprogramm zeigt einen einfachen endlichen Automaten, der seinen Zustand ändert wenn man die Bumper drückt. Was das Programm genau tut, sieht man am Besten, wenn man es einfach mal ausführt und je zweimal auf die Bumper drückt. Dabei sollte man sich die Ausgaben im Terminal und die Status LEDs anschauen und die Bumper zunächst langsam drücken und wieder loslassen!

Meistens werden endliche Automaten in C mit einem switch-case Konstrukt realisiert (oder einem Haufen if-else-if-else Bedingungen... switch ist jedoch übersichtlicher).

Ein kleines Beispiel:

```
#include "RP6RobotBaseLib.h"

#define STATE_START   0
#define STATE_1       1
#define STATE_2       2
#define STATE_3       3

uint8_t state = STATE_START;

void simple_stateMachine(void)
{
   switch(state)
   {
      case STATE_START: writeString("\nSTART\n");     state = STATE_1;
        break;
      case STATE_1:     writeString("Zustand 1\n");   state = STATE_2;
        break;
      case STATE_2:     writeString("Zustand 2\n");   state = STATE_3;
        break;
      case STATE_3:     writeString("Zustand 3\n");   state = STATE_START;
        break;
   }
}

int main(void)
{
   initRobotBase();
   while(true)
   {
       simple_stateMachine();
       mSleep(500);
   }
   return 0;
}
```

Das hier ist nur ein sehr sehr simples Beispiel um das Prinzip zu verdeutlichen. Ein Automat besteht aus Zuständen und Übergängen zwischen diesen Zuständen. Unsere Zustände sind hier STATE_START und STATE_1 bis 3. Den Automaten im obigen Listing könnte man auch folgendermaßen als Graphen darstellen:

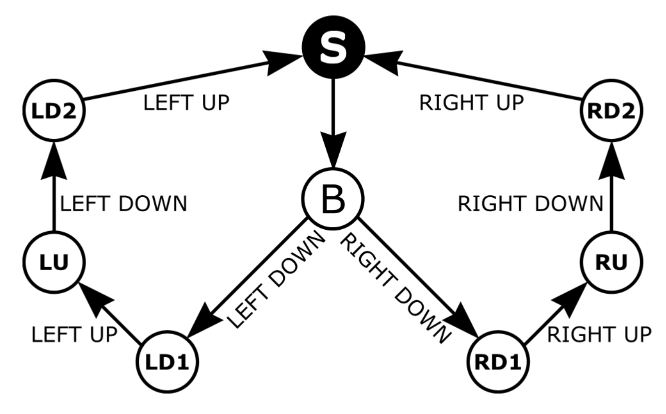


S ist dabei der Startzustand. Da wir hier keine Bedingungen für den Wechsel in einen anderen Zustand eingebaut haben, wechselt der Automat in jedem Schritt in den nächsten Zustand und in Zustand 3 wieder zurück in den Startzustand. Zwischen den einzelnen Schritten haben wir in diesem einfachen Programm daher je 500ms Pause eingefügt, damit die Ausgaben nicht zu schnell erfolgen.
Die Ausgabe des Programms wäre folgendes:

```
START
Zustand 1
Zustand 2
Zustand 3

START
Zustand 1
Zustand 2
...
... usw.
```

Das würde das Programm nun beliebig lange ausgeben.

Das Beispielprogramm in der Datei **RP6Base_Move_04_FSM.c** hat allerdings einen etwas komplexeren Automaten mit 8 Zuständen. Die grundlegende Struktur ist in folgendem Graphen dargestellt (die Zustände heissen im Programm natürlich anders, ist hier um Platz zu sparen abgekürzt worden):



Der Automat startet im Zustand S und wechselt sofort (nachdem er eine kleine Textnachricht ausgegeben hat) in den Zustand B. Dort wartet er, bis einer der beiden Bumper gedrückt worden ist und wechselt dann entweder in den Zustand LD1, wenn es der linke Bumper war, oder in den Zustand RD1, beim rechten Bumper. Dann wird gewartet bis der Bumper wieder losgelassen wird und jeweils in den Zustand LU oder RU gewechselt. Dort wird dann auf den zweiten Druck auf den jeweiligen Bumper gewartet. In diesem Zustand und auch schon in dem davor interessiert es übrigens nicht, was mit dem anderen Bumper passiert – da kann man drauf rumdrücken wie man will, darauf wird nicht reagiert. Drückt man den Bumper nun ein weiteres mal, wird in den Zustand RD2 bzw. LD2 gewechselt und lässt man den Bumper dann wieder los, wird wieder zurück in den Startzustand gewechselt und alles beginnt von vorn.
Natürlich gibt der Automat im Beispielprogramm bei jeder Zustandsänderung auch

Text auf der seriellen Schnittstelle aus und setzt die Status LEDs, aber dafür ist kein Platz mehr in dem Graphen oben gewesen. Der obige Graph soll auch nur den grundlegenden Aufbau verdeutlichen.

Beispiel 10: Endliche Automaten Teil 2
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_05_Move_04_FSM2
Datei: RP6Base_Move_04_FSM2.c

ACHTUNG: Der Roboter bewegt sich in diesem Beispielprogramm!

Nun probieren wir uns mal an einem Automaten bei dem sich der Roboter auch bewegt! Die Funktion des Programms ist einfach gehalten – zunächst blinkt Status LED5. Damit will uns der Roboter signalisieren: „Drück mal bitte jemand auf den linken Bumper!“. Tut man das, fährt der RP6 etwa 15cm rückwärts und lässt dann die Status LED2 blinken – was bedeuten soll, dass nun der rechte Bumper gedrückt werden muss, damit etwas passiert. Tut man auch das, fährt der RP6 die 15cm wieder vorwärts und dieser Zyklus beginnt von neuem.

Die Struktur des Automaten ist eigentlich nichts besonderes – fast genau wie der Automat von oben der ständig „START Zustand1 Zustand2...“ ausgegeben hat. Nur haben wir nun ein paar weitere Bedingungen eingebaut. Beispielsweise warten wir in zwei Zuständen mit der „isMovementComplete()“ Funktion darauf, dass der Roboter den Bewegungsvorgang abschließt. Und noch eine kleine Besonderheit: wir führen in zweien der Zustände ein einfaches Programmfragment aus. In diesem Fall benutzten wir eine Stopwatch dazu eine LED alle 500ms umzuschalten. Dort könnte auch etwas ganz anderes ausgeführt werden – z.B. das Lauflicht aus Beispiel 1.

Man könnte jetzt noch viele Seiten über Automaten und ähnliches füllen, aber wie gesagt – das hier ist eine Bedienungsanleitung und kein Fachbuch. Also weiter mit den Beispielprogrammen...

Beispiel 11: Verhaltensbasierter Roboter
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_05_Move_04
Datei: RP6Base_Move_04.c

ACHTUNG: Der Roboter bewegt sich in diesem Beispielprogramm!

Worauf wir mit den Beispielen zu Automaten eigentlich hingearbeitet haben, findet sich in diesem Beispielprogramm.

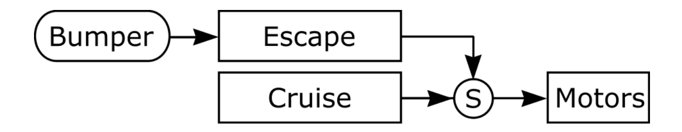
Hier wurde ein einfacher, verhaltensbasierter Roboter implementiert. Der Roboter hat hier zwar nur zwei kleine Verhaltensweisen, aber es soll ja zunächst nicht zu kompliziert werden. Wir basteln uns in den nächsten Programmen dann Schritt für Schritt ein einfaches insektenähnliches Verhalten zusammen. Jetzt im ersten Schritt hat unser „Insekt“ allerdings nur zwei kleine Fühler, die melden ob es eine Kollision gibt oder nicht.

Die beiden Verhaltensweisen auf die wir uns zunächst beschränken werden, heissen „Cruise“ und „Escape“. Das Verhalten Cruise gibt einfach nur den Befehl „fahre Vorwärts“. Es tut sonst nichts weiter. Da könnte man natürlich noch ein wenig mehr einbauen und auch z.B. zeitgesteuert mal eine Kurve fahren, langsamer und schneller

werden je nach Akkuzustand etc. pp.. Wenn man das Programm allerdings modular implementieren will, sollte man sowas auch in zusätzliche Verhalten auslagern.

Escape ist dagegen schon relativ komplex und wird bei Zusammenstößen mit Objekten die von den Bumpern gemeldet werden aktiv. Dann wird - je nachdem welche Bumper getroffen wurden – ein paar Zentimeter zurückgesetzt, etwas gedreht und anschließend die Kontrolle wieder an das andere Verhalten übergeben.

In einem Diagramm könnte man das wie folgt darstellen:



Das Verhalten Escape unterdrückt die Ausgaben des Verhaltens Cruise, über den „Suppressor" S und gibt seine eigenen Befehle an die Motoren weiter. Escape hat also die höhere Priorität der beiden Verhalten.

Schon mit diesen beiden einfachen Verhalten fährt der Roboter umher und „erkundet" seine Umgebung. Er kann dabei nur auf zwei Tastsensoren zurückgreifen, die nur melden, ob ein Objekt detektiert wurde oder nicht. Das lässt natürlich nicht gerade besonders komplexe Verhaltensweisen zu.

Stellen Sie sich einfach mal vor, Sie müssten sich in Ihrer Wohnung nur mit zwei Fingerspitzen zurechtfinden – keine Augen, keine Ohren, keine sonstigen Tastsensoren – nur die beiden Fingerspitzen, die Sie starr nach vorne halten können...

Je mehr Sensoren wir dem Roboter zur Verfügung stellen, desto komplexer können auch seine Verhaltensweisen werden! Ein typisches Insekt hat beispielsweise noch weit mehr Tastsensoren als nur zwei, die zudem auch noch analoge Werte liefern die etwas darüber aussagen wie stark der Druck ist. Man kann sich leicht vorstellen das der Roboter mit den Facettenaugen von typischen Insekten auch schon um einiges „intelligenter" handeln könnte...

Wir nutzen hier ein Verfahren, welches von Rodney Brooks (`http://people.csail.mit.edu/brooks/`) bereits um 1985 entwickelt wurde. Die sog. Subsumptions Architektur.

Das englischsprachige Original findet man hier:

`http://people.csail.mit.edu/brooks/papers/AIM-864.pdf`

Aber es gibt beispielsweise (einfach mal im Internet danach suchen!) hier:

`http://www.igi.tugraz.at/STIB/WS98/Bergmann/einleitung.htm`

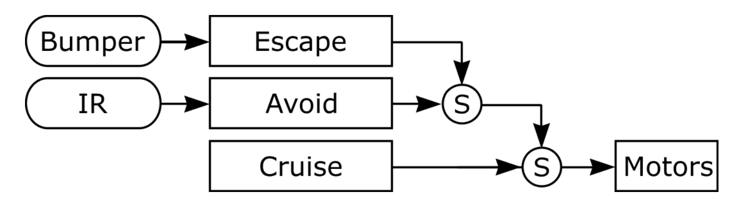
auch eine kurze deutsche Zusammenfassung davon.

(Es gibt noch viel mehr Doku dazu im Internet – einfach mal ein wenig suchen!)

Beispiel 12: Verhaltensbasierter Roboter 2
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_05_Move_05
Datei: RP6Base_Move_05.c

ACHTUNG: Der Roboter bewegt sich in diesem Beispielprogramm!

In diesem Beispiel wurde unser verhaltensbasierter Roboter um ein Verhalten namens „Avoid" erweitert. Mit diesem kann das ACS mit eingebunden werden und so nicht nur bei Kollisionen reagiert werden, sondern schon vorher versucht werden Hindernissen auszuweichen. Die drei Verhalten haben nun also folgende Struktur:



Das Verhalten Escape hat noch immer die höchste Priorität, denn wenn der Roboter mit einem Hindernis zusammenstößt, ist diese Information natürlich wichtiger als der Zustand des ACS. Escape unterdrückt also die Befehle von Avoid und Avoid tut dies wiederum mit den Befehlen von Cruise.

Detektieren die IR Sensoren ein Hindernis, werden vom Avoid Verhalten Ausweichmanöver eingeleitet. Detektiert der linke ACS Kanal ein Hindernis, wird eine Kurve nach Rechts gefahren und beim rechten ACS Kanal entsprechend eine Rechtskurve. Wenn beide Kanäle ein Hindernis melden, wird je nachdem welcher Kanal zuvor aktiv war, auf der Stelle nach links oder rechts rotiert. Der Roboter dreht sich auch nachdem beide Kanäle wieder freie fahrt melden noch etwas weiter. Das wird über Stopwatches gesteuert.

Beispiel 13: LDRs - Lichtsensoren
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_06_LightDetection
Datei: RP6Base_LightDetection.c

Das Programm erzeugt Ausgaben auf der seriellen Schnittstelle

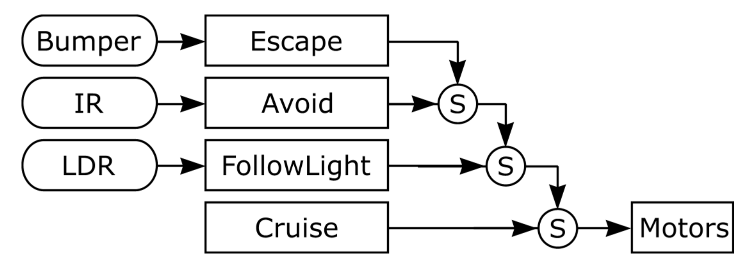
Der Roboter bewegt sich in diesem Beispielprogramm *nicht*!

Dieses Beispielprogramm demonstriert die Verwendung der Lichtsensoren. Es zeigt mit den StatusLEDs an, welcher Lichtsensor stärker als der andere beleuchtet wird, oder ob beide in etwa gleich stark beleuchtet werden. Das wird auch kontinuierlich zusammen mit den Messwerten auf der seriellen Schnittstelle ausgegeben.

Beispiel 14: Verhaltensbasierter Roboter 3
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_07_LightFollowing
Datei: RP6Base_LightFollowing.c

ACHTUNG: Der Roboter bewegt sich in diesem Beispielprogramm!

Die Lichtsensoren können wir natürlich ebenfalls über ein kleines Verhalten namens „FollowLight" mit einbeziehen:



Die Priorität von FollowLight liegt unterhalb von Escape und Avoid, aber oberhalb von Cruise. Daher ist Cruise in diesem Beispiel eigentlich nie aktiv. Es sei denn, es wird zu dunkel (LDR Werte beide unter 100), dann wird das Verhalten FollowLight deaktiviert.

FollowLight versucht mit den LDRs hellen Lichtquellen zu folgen bzw. die hellste Stelle im Raum zu finden. Das klappt natürlich nicht immer optimal – z.B. wenn es viele verschiedene Lichtquellen gibt. Wenn man das aber in einem dunkeln Raum mit einem starken Handschweinwerfer testet, sollte das schon ziemlich gut klappen.

Die LEDs werden in diesem Programm doppelt verwendet – wenn das ACS kein Hindernis erkannt hat, wird angezeigt welcher Lichtsensor die höhere Lichtintensität detektiert hat und wenn das ACS ein Hindernis entdeckt, wird das mit den LEDs angezeigt.

Das war es dann auch mit den Beispielprogrammen zu verhaltensgesteuerten Robotern. Insgesamt hat er nun das Verhalten eines einfachen Insekts wie z.B. einer Motte, die das Licht sucht und dabei Hindernissen ausweicht.

Nun könnte man für zusätzliche Sensoren auf Erweiterungsmodulen eigene Verhaltensmuster einbauen und die bestehenden noch stark verbessern. Hier sind jetzt eigene Kreativität und Programmierfähigkeiten gefragt!

Beispiel 15: Fernsteuerung mit einer RC5 Fernbedienung
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_08_TV_REMOTE
Datei: RP6Base_TV_REMOTE.c

ACHTUNG: Der Roboter bewegt sich in diesem Beispielprogramm!

Dieses Beispielprogramm ermöglicht es, den Roboter über eine RC5 Fernbedienung ähnlich wie ein ferngesteuertes Auto zu steuern. Nur gibt es beim RP6 noch ein paar andere Bewegungsmöglichkeiten, als bei den meisten ferngesteuerten Autos. Er kann natürlich vorwärts und rückwärts fahren, aber auch auf der Stelle nach links oder rechts drehen. Dann kann er Kurven nach vorne links und rechts, bzw. hinten links und rechts fahren und außerdem jeweils nur einen der beiden Motoren anschalten und vorwärts oder rückwärts laufen lassen.

In diesem Programm ist das so realisert, dass die verschiedenen Bewegungen bei Empfang eines vorher zugeordneten Tastencodes von der Fernbedienung gestartet werden und bei dauerhaft gedrückter Taste langsam beschleunigt wird. Lässt man die Taste los, wird - noch etwas langsamer als bei der Beschleunigung – wieder abge-

bremst. Der Roboter beschleunigt und bremst also absichtlich so langsam. Man kann die Bewegung mit einer Taste auch abrupt stoppen – dann bremst der Roboter sofort ab und kommt (ohne besonders viel Last jedenfalls und abhängig von der Geschwindigkeit) innerhalb von etwa ein bis zwei Zentimetern zum Stehen.

Der RC5 Empfang kann natürlich noch für viele andere Dinge verwendet werden – z.B. verschiedene Programme starten, Verhaltensweisen beeinflussen, Regelungsparameter der Geschwindigkeitsregelung verändern etc. pp..

Es wäre auch möglich über die Adressierung mehrere Roboter mit einer einzigen universal Fernbedienung zu steuern (die haben meist mehrere Wahltasten für die verschiedenen Geräte – die muss man so programmieren, dass sie alle im RC5 Code senden und unterschiedliche Adressen haben – sofern das mit der jeweiligen Fernbedienung möglich ist).

Beispiel 16: I^{2}C Bus Interface – Master Modus
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_I2C_MASTER_01
Datei: RP6Base_I2C_MASTER_01.c

Dieses Programm demonstriert wie man den Master Modus des I^{2}C Busses verwenden kann. Man benötigt natürlich einen passenden Slave der am I^{2}C Bus angeschlossen ist. Wie man diesen ansteuert, variiert von Slave zu Slave.

In diesem Beispiel wird mit 8 LEDs ein kleines „Knight Rider" Lauflicht realisiert. Die LEDs sind an einen PCF8574 angeschlossen. Der PCF8574 ist ein gängiger 8-Bit Portexpander mit I^{2}C Interface. Man kann einen PCF8574 z.B. auf eine der Experimentier Erweiterungsplatinen löten (oder erstmal zum Testen auf einem Steckbrett aufbauen) und schon hat man 8 freie I/O Ports mit denen man digitale Sensoren auswerten und über externe Transistoren oder Treiberbausteine auch Lasten schalten kann. LEDs und andere sehr kleine Verbraucher kann man direkt anschließen.

Ein sehr praktischer Chip also! Man kann auch mehrere davon verwenden – dann muss man nur die Adresse des Chips mit den drei Adresspins verändern bzw. bei mehr als 8 Portexpandern zusätzlich den PCF8574A verwenden – der hat einen anderen Adressraum, und so kann man theoretisch je 8 dieser beiden ICs, also 16*8 = 128 I/O Port Pins am I^{2}C Bus betreiben!

Beispiel 17: I^{2}C Bus Interface – Master Modus
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_I2C_MASTER_02
Datei: RP6Base_I2C_MASTER_02.c

Dieses Programm demonstriert wie man den Master Modus des I^{2}C Busses verwenden kann. Man benötigt natürlich einen passenden Slave der am I^{2}C Bus angeschlossen ist.

Jetzt kommen noch Routinen für einen PCF8591 zu unserem Programm hinzu. Dieser Chip bietet einen 8-Bit Analog/Digital Wandler (ADC) mit 4 Kanälen und einen Digital/Analog Wandler (DAC), mit dem man eine analoge Spannung ausgeben kann. Der PCF8574 und der PCF8591 ergänzen sich so sehr gut. Hier können vier beliebige Spannungen gemessen werden – in diesem Beispiel nehmen wir an, dass vier zusätzliche LDRs am PCF8591 angeschlossen sind (als Spannungsteiler geschaltet!). Das ist aber eher nebensächlich – es könnten auch genausogut vier GP2D120 IR Abstandssensoren von Sharp, oder Temperatursensoren oder etwas ganz anderes sein.

Einziger Nachteil dieser beiden ICs ist natürlich, dass die Auswertung über den I^{2}C Bus relativ viel Zeit in Anspruch nimmt. Die beiden ICs sind daher nur für einfache Aufgaben gedacht. Wenn man auf sehr schnelle Auswertung und Steuerung angewiesen ist, kommt man um einen weiteren Mikrocontroller meist nicht herum. Das ist

dann zwar etwas komplizierter zu handhaben, aber auch deutlich flexibler, da man den Mikrocontroller beliebig programmieren kann!

Die Datenblätter vom PCF8574 und PCF8591 mit genauen Infos finden Sie auf der CD!

Beispiel 18: I²C Bus Interface – Master Modus
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_I2C_MASTER_03
Datei: RP6Base_I2C_MASTER_03.c

Dieses Programm demonstriert wie man den Master Modus des I²C Busses verwenden kann. Man benötigt natürlich einen passenden Slave der am I²C Bus angeschlossen ist.

Ein kleines Demoprogramm wie man einen Devantech SRF08 oder SRF10 Ultraschallsensor über den I²C Bus ansteuern kann. Man kann das natürlich auch für andere Sensoren anderer Hersteller umschreiben.

Beispiel 19: I²C Bus Interface – Master Modus
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_I2C_MASTER_04
Datei: RP6Base_I2C_MASTER_04.c

Dieses Programm demonstriert wie man den Master Modus des I²C Busses verwenden kann. Man benötigt natürlich einen passenden Slave der am I²C Bus angeschlossen ist.

In diesem Programm werden vier I²C Busteilnehmer angesteuert. Zwei SRF08/SRF10, sowie ein PCF8574 und ein PCF8591. Es wurden die Programmteile der drei vorherigen Beispielprogramme verwendet.

Weitere Beispielprogramme zu I²C Peripherie werden zusammen mit den Erweiterungsmodule veröffentlicht!

Beispiel 20: I²C Bus Interface – Slave Modus
Verzeichnis: <RP6Examples>\RP6BaseExamples\Example_I2C_SLAVE
Datei: RP6Base_I2C_SLAVE.c

Dieses Programm tut von sich aus erstmal nichts. Man muss den Roboter mit einem Erweiterungsmodul ausstatten, welches die Kontrolle übernimmt und als I²C Master agiert.

Es ist sehr sinnvoll, den Roboter mit weiteren Controllern auszustatten. Meist möchte man dann auch gleich einen der Controller auf den Erweiterungsmodulen zur Steuerung des ganzen Roboters verwenden. Das ist deshalb sinnvoll, weil der Controller auf der Roboterbasiseinheit schon relativ viel mit dem ACS, der Motorregelung und allem anderen zu tun hat. Auf einem externen Controller steht noch viel mehr Rechenzeit zur Verfügung, die auch nicht von diesen anderen Aufgaben unterbrochen wird.

Genau dafür ist dieses Programm gedacht: Alle wichtigen Funktionen des Roboters kann man mit diesem Programm über den I²C Bus steuern. Der Controller ist als Slave Device unter einer einstellbaren Adresse ansprechbar. Die bekannten Features, wie die automatische Geschwindigkeitsregelung kommen mit diesem Programm besonders gut zur Geltung, da man jetzt nämlich mit einem Master Controller z.B. einfach nur kurz das Kommando „Fahre mit <Geschwindigkeit> geradeaus“ über den Bus senden muss und der Rest dann automatisch erledigt wird, ohne dass man sich weiter drum kümmern müsste. Natürlich kann man diesen Vorgang auch unterbrechen, z.B. wenn ein Sensor ein Hindernis meldet.

Im Programmcode ist eine Liste mit den gültigen Befehlen und der zugehörigen Parameter zu finden. Dort finden Sie auch weitere Informationen zur Ansteuerung.

Der Controller auf der Roboterbasis löst bei Bedarf automatisch ein Interrupt Signal auf der ersten Interruptleitung (INT1) der Erweiterungsanschlüsse aus, wenn sich der Zustand der Sensoren geändert hat, oder ein Fahrbefehl ausgeführt wurde.

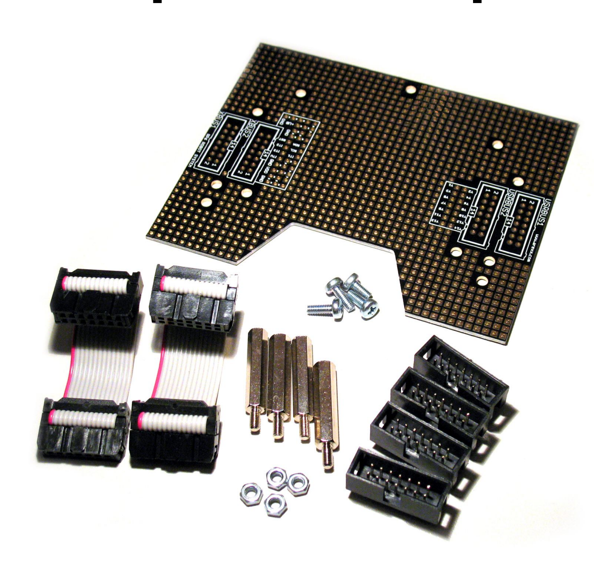
Alle Sensorwerte können über eine Reihe von Registern ausgelesen werden – wenn also ein Interrupt ausgelöst wurde, kann man erst einige Statusregister auslesen und überprüfen, welcher Sensor das Ereignis ausgelöst hat. Dann kann man anschließend die betreffenden Register auslesen.

Alternativ kann man auch einfach alle Sensorregister in einem Rutsch auslesen. Das kostet natürlich etwas Zeit.

Mehr Details dazu und vor allem die genauen Registerbezeichnungen finden Sie im Quellcode des Programms.

Weitere Beispielprogramme zur Ansteuerung des Slaves werden demnächst zusammen mit den Erweiterungsmodulen veröffentlicht. Einige sind bereits auf der CD-ROM zusammen mit den Beispielen für das RP6 CONTROL M32 zu finden.

5. Experimentierplatine



Im Lieferumfang des Roboters befindet sich eine Experimentierplatine. Darauf können Sie eigene Schaltungen mit bedrahteten Bauelementen aufbauen. Die Platine kommt als Bausatz – man muss noch die Stecker RICHTIG HERUM (s. Platinenaufdruck) anlöten bevor man etwas damit anfangen kann.

Um dann Schaltungen darauf aufbauen zu können, braucht man natürlich ein wenig Löterfahrung und muss in etwa wissen was man da eigentlich tut.

Was kann man denn für Schaltungen auf so einer Platine aufbauen?

Oben haben wir schon zwei Beispiele gegeben! Einmal könnte man dem Roboter mit I²C Portexpandern oder A/D Wandlern zusätzliche I/O Ports und A/D Kanäle spendieren und daran Sensoren oder Aktoren anschließen (Lichtsensoren, IR Sensoren, Tastsensoren, LEDs, etc. pp.). Und dann gibt es natürlich noch die Möglichkeit, komplexere Sensormodule mit I²C-Bus Interface daran anzuschließen. So z.B. Ultraschallsensoren, elektronische Kompassmodule, Drehraten- und Beschleunigungssensoren. Interessant wären auch Druck-, Temperatur- und Luftfeuchtigkeitssensoren um eine mobilen Wetterstation zu bauen.

Man kann übrigens beliebig viele dieser Erweiterungsplatinen auf den Roboter schrauben. Es muss nicht bei der einen bleiben. Wenn man komplexere Dinge damit anstellen will, muss man einen zusätzlichen Mikrocontroller verwenden. Den gibt es z.B. auf dem RP6 Control M32, das auch noch über 14 freie I/Os verfügt, wovon 6 A/D Wandler sind. Die I/Os sind auf 10 polige Anschlüsse herausgeführt. Man kann sich dafür einfach kleine Flachbandkabel basteln und das Modul darüber mit der Erweiterungsplatine verbinden.

Auch das demnächst erhältliche C-Control Erweiterungsmodul wird sich dafür eignen.

Auf dem Mainboard des Roboters gibt es natürlich auch noch die 6 Erweiterungsflächen. Wenn man Sensoren so tief wie möglich anbringen möchte (z.B. weitere IR-Sensoren) ist das recht praktisch. Aber bevor Sie am Mainboard herumlöten, sollten Sie lieber erstmal mit der normalen Experimentierplatine anfangen – die kann man leichter auswechseln wenn mal etwas schief geht...

6. Schlusswort

Hier endet dieser kleine Einstieg in die Welt der Robotik! Wir hoffen, dass Sie schon viel Spaß mit dem RP6 hatten und auch weiterhin haben werden!

Aller Anfang kann schwer sein, gerade bei dieser komplexen Materie, aber wenn man die ersten Hürden genommen hat, macht die Robotik sehr viel Spaß und man lernt auch viel dabei.

Wenn man sich später ein wenig besser in der C Programmierung auskennt, kann man richtig loslegen und größere Ideen realisieren. Es wurden in dieser Anleitung ja schon ein paar Beispiele gegeben, was man mit dem Roboter alles machen könnte. Natürlich könnten Ihnen noch ganz andere interessante Dinge einfallen!

Wie schon oft erwähnt wurde, kann man sich mit anderen Anwendern über die diversen Internet Foren die sich mit Robotik und Elektronik beschäftigen sehr gut austauschen und sich gegenseitig bei Problemen helfen. Das kann oft viel hilfreicher sein, als jede noch so ausführliche Anleitung, zumal wir hier noch längst nicht alle Themen besprochen haben und vieles nur sehr knapp dargestellt werden konnte.

Es sind natürlich noch einige Erweiterungsmodule für den RP6 geplant. Bereits verfügbar sind das Lochraster Erweiterungsmodul und das RP6 CONTROL M32 mit weiterem MEGA32 Controller (+ Mikrofonsensor, Piezo Schallwandler, großem externen 32KB EEPROM Speicher, Tastern, LEDs und einem LCD Port). Geplant ist ein Erweiterungsmodul für neuere Conrad Electronic C-Control Module wie z.B. das C-Control PRO MEGA128.

Es werden auch Module mit zusätzlichen Sensoren entwickelt. Mehr können wir an dieser Stelle allerdings noch nicht verraten...

Wenn weitere Module verfügbar sind, wird das aber auf der AREXX Homepage und im Forum bekannt gegeben!

Bis dahin haben Sie aber sicherlich auch so schon genug mit dem RP6 und den bereits verfügbaren Erweiterungen zu tun!

ANHANG

A - Fehlersuche

Hier eine kleine Liste mit Fehlern und möglichen Ursachen bzw. Lösungsvorschlägen. Die Liste wird in Zukunft noch erweitert und aktualisiert werden!

1. **Der Roboter lässt sich nicht einschalten – es leuchtet keine der LEDs und er reagiert auch nicht auf den Start/Stop Taster!**
   - Das USB Interface ist nur am Roboter angeschlossen, aber nicht am PC! Dann wird der Mikrocontroller auf dem Roboter im Reset Modus gehalten und kann daher auch keine der LEDs ansteuern! Schließen Sie das USB Interface immer zuerst am PC an bevor Sie es mit dem RP6 verbinden! Wenn der PC ausgeschaltet ist, kann das übrigens auch passieren!
   - Ein Erweiterungsmodul hält alle Mikrocontroller im Reset Modus – das kann bei einer eigenen Konstruktion schonmal passieren (auch bei fertigen Modulen, dann aber eigentlich nur, wenn diese defekt oder falsch programmiert sind). Sie können das testen indem Sie alle Erweiterungsmodule entfernen!
   - Es sind keine Akkus eingebaut oder die Akkus sind leer.
   - Die Sicherung ist durchgebrannt. Das kann man mit einem Durchgangsprüfer testen (z.B. mit einem Multimeter) und oft auch einfach so erkennen indem man sich den Draht innerhalb des Glaszylinders der Sicherung anschaut – oft brennt dieser in der Mitte durch (aber nicht immer! Deshalb muss man auch wenn nichts zu erkennen ist, zur Sicherheit mit einem Multimeter o.ä. nachprüfen!).

     ACHTUNG: Wenn die Sicherung durchbrennt, heisst das normalerweise, dass irgendetwas überhaupt nicht stimmt! Das passiert nämlich entweder, weil irgendwo ein Kurzschluss verursacht wurde (haben Sie evtl. versehentlich irgendwelche Metallteile auf den Roboter gelegt?) oder weil ein oder mehrere Bauteile zuviel Strom benötigt haben. Schauen Sie sich in diesem Fall auch mal die Platine und alle Bauteile etwas genauer an! Haben Sie evtl. irgendwelche Veränderungen am Roboter vorgenommen oder sind offensichtlich defekte Bauteile zu erkennen? Sind die Akkus Polungsrichtig in den Akkuhalter eingelegt worden? Dabei keine Kabel zwischen Mainboard und Chassis eingequetscht? Haben Sie kürzlich Erweiterungsmodule hinzugefügt? Falls ja, alle vor einem erneuten Versuch abmontieren! Wenn scheinbar alles OK ist und alle Erweiterungsmodule abmontiert sind, können Sie eine passende, flinke 2.5A Feinsicherung einbauen und es nochmal probieren.
2. **Der Roboter startet die Programme nicht!**
   - Haben Sie nach dem Programmupload auch auf den Start/Stop Taster gedrückt?
   - Haben Sie ein Programm korrekt in den RP6 geladen? Ist es das richtige Programm?

3. **Der Roboter löst während der Fahrt ständig Resets aus und stoppt die Programmausführung!**
   - Die Akkus sind nicht ausreichend geladen.
   - Die verwendeten Akkus sind von schlechter Qualität oder schon sehr alt und daher bricht die Spannung beim Betrieb zu stark ein, was dann automatisch einen Reset auslöst. Verwenden Sie besser neue Akkus für den Roboter!
   - Die Akkus sitzen locker im Akkuhalter oder haben aus sonstigen Gründen nur wackligen Kontakt zum Akkuhalter.
   - Ein Erweiterungsmodul o.ä. könnte kurzzeitig zuviel Energie benötigen!
4. **Der RP6Loader kann keine Verbindung zum Roboter aufbauen!**
   - Das USB Kabel oder das 10polige Flachbandkabel sind nicht richtig angeschlossen (auch mal am USB Stecker wackeln!).
   - Sie haben den falschen Port in der RP6Loader Software ausgewählt.
   - Der Port wird von einer anderen Anwendung verwendet und taucht deshalb nicht in der Liste im RP6Loader auf. Beenden Sie in diesem Fall alle Anwendungen die evtl. auf USB Ports bzw. USB-Seriell Wandler zugreifen könnten und aktualisieren Sie die Portliste im RP6Loader danach oder starten Sie den RP6Loader neu! (s.a. 5)
   - Sie haben in den Einstellungen vom RP6Loader „Reset Pin invertieren" eingestellt – deaktivieren Sie diese Option!
   - Der Roboter ist ausgeschaltet oder die Akkus sind leer bzw. fast leer.
   - Das Kabel oder die Stecker/Buchsen sind beschädigt – das ist zwar unwahrscheinlich, kann aber vorkommen wenn man diese zu stark belastet.
5. **Im RP6Loader werden keine Ports angezeigt!**
   - Überprüfen Sie ob die Ports von einer anderen Anwendung verwendet werden. Die Ports werden nur angezeigt, wenn Sie auch verfügbar sind (s.o.).
   - Wenn Sie Ubuntu 7.x verwenden, können Sie versuchen die Pakete „brltty" und „brltty-X11" mit Synaptic zu deinstallieren. „brltty" ist ein Programm das bei Ubuntu standardmäßig installiert ist und eigentlich für sog. „Braille" Displays gedacht ist (Blindenschrift „Displays"). Leider übernimmt dieses Programm oft automatisch neu angeschlossene Ports und verhindert somit, dass der RP6Loader darauf zugreifen kann.
   - Bei anderen Linux Systemen müssen Sie evtl. Ihren Benutzeraccount zu den Gruppen „uucp" und/oder „lock" hinzufügen. Dazu können Sie entweder die jeweiligen Tools Ihrer Distribution verwenden, oder per Hand die Datei „/etc/groups" bzw. „/etc/group" editieren. Sie benötigen dazu root Rechte. Tragen Sie Ihren Benutzernamen in dieser Datei bei den Einträgen „lock:x:53:username" und „uucp:x:14:username" anstelle von „username" ein oder fügen Sie ihn durch Komma getrennt hinten an. Wenn keiner der Einträge vorhanden ist, fügen Sie einfach beide neu hinzu. Dann müssen Sie noch als root die schreib/lese Rechte für den Port setzen. Also „chmod 777 /dev/ttyUSB0" ausführen sofern dem USB Interface der Port /dev/ttyUSB0 zugewiesen wurde. Danach einmal neu anmelden oder den Rechner neustarten. Nun sollte es funktionieren.

6. Der Roboter gibt während der Fahrt seltsame Geräusche von sich!

- Das kann verschiedene Ursachen haben. Sie müssen den Roboter auf jeden Fall aufschrauben und sich die Getriebe und Motoren ansehen. Evtl. sind beim Einbau der Akkus Kabel in die Getriebe geraten? **Ein paar Klackergeräusche sind normal und kein Grund zur Sorge!** Der Roboter wird bei höheren Geschwindigkeiten natürlich auch deutlich lauter (aber kein Vergleich zum extrem lauten Vorgänger CCRP5).
- Hat sich einer der Stellringe (Befestigungsschrauben) an den Zahnrädern oder den Antriebsrädern gelockert oder sitzt sonst etwas verdächtig lose? Alles muss sich frei drehen können (mal vorsichtig an den Antriebsrädern drehen!), aber die Zahnräder sollten nicht aneinander schleifen oder ähnliches!
- Ein oder zwei Tropfen Lageröl an die beiden Lager jedes Motors - also vorne und hinten wo man die Motorwelle sehen kann - kann die Geräuschentwicklung schon deutlich reduzieren (**ACHTUNG:** Nur spezielles Lageröl verwenden!!! Beispielsweise Conrad Artikelnummer 223441-62 „Team Orion Free Revs Kugellageröl Nr. 41701")! Nachdem und während man etwas Öl auf die Lager träufelt, sollte man den Motor hin und her drehen damit sich das Öl gut im Lager verteilt (den Motor dabei ein paar Sekunden hochkant halten damit es in das Lager laufen kann)! Rückstände die noch aussen herum vorhanden sind, sollte man wegwischen nachdem einiges von dem Öl in das Lager geflossen ist! Werksseitig sollten die Lager zwar bereits geölt worden sein, aber nach einer gewissen Zeit muss man die Lager evtl. neu ölen!

  Aber bitte ***nicht*** auf die Idee kommen die Zahnräder damit zu ölen! Das wird die Drehgeber mit ziemlicher Sicherheit beschädigen bzw. funktionsunfähig machen! Nicht unbedingt die Elektronik, aber die Codescheiben/Aufkleber, die jede Flüssigkeit gut aufsaugen können, was die Reflexionseigenschaften drastisch verändert! Außerdem kann flüssiges Lageröl oder sonstiges flüssiges Öl/Schmiermittel auch im ganzen Roboter umherspritzen wenn sich die Zahnräder drehen! **Schmierfett (also nicht flüssig!) gehört *nur* auf die Wellen (!) von den beiden Stufenrädern!** Fett bringt normalerweise auch keine signifikante Verbesserung...

7. Die Motoren beschleunigen bei jedem Programm immer nur ganz kurz auf eine sehr hohe Geschwindigkeit, stoppen dann sofort und die vier roten LEDs fangen an zu blinken!

- Die einfachste Möglichkeit: Sie haben ein eigenes Programm oder ein geändertes Beispielprogramm in den Roboter geladen und vergessen den Befehl „`powerON();`" auszuführen bevor Sie die Motoren starten!
- Haben Sie die Software sonst irgendwie geändert? Funktionieren andere Beispielprogramme bzw. der Selbsttest?
- Falls das nicht zutrifft: es sollte auch zusätzlich zum Blinken der LEDs eine Fehlermeldung auf der seriellen Schnittstelle ausgegeben werden, die das Problem näher beschreibt! Das deutet meistens darauf hin, dass etwas mit den Encodern nicht stimmt. Vielleicht haben sich wie oben auch schon beschrieben die Stellringe, die normalerweise die beiden Stufenräder mit den

Codescheiben an Ort und Stelle halten gelockert? Dann können die Zahnräder sich zu weit von den Sensoren entfernen und diese liefern dann kein Signal mehr! Auf den Encodern ist ein kleines Potentiometer vorhanden (=einstellbarer Widerstand). Damit kann man die Sensoren justieren (feinen 1.8 bis 2mm Schlitz Schraubendreher oder passenden Kreuzschlitz Schraubendreher verwenden! **Zum Einstellen der Encoder gibt es ein separates PDF Dokument, das die Einstellung sehr detailliert beschreibt! Sie finden es als Download auf der RP6 Homepage unter `http://www.arexx.com/rp6/` !**

- Sind die Codescheiben evtl. beschädigt worden? Beispielsweise durch Öl oder Schmierfett wie oben beschrieben?
- Sind die Kabel zu den Encodern in Ordnung oder evtl. beschädigt? Am Besten mit einem Multimeter überprüfen (Durchgangsprüfer bei ausgeschaltetem Roboter jeweils an die Kabelenden halten!)!
- Sind die kleinen Reflexlichtschranken der Encoder verschmutzt?
- Die LEDs fangen auch an zu blinken wenn einer oder beide Motoren nicht korrekt funktionieren – also Kabel überprüfen und die Platine im Bereich der Motorelektronik etwas genauer ansehen.

8. Der Roboter bewegt sich nicht oder nur langsam und/oder nach kurzer Zeit bzw. sofort nach Programmstart fangen die vier roten LEDs an zu blinken!

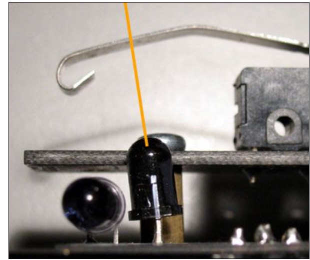
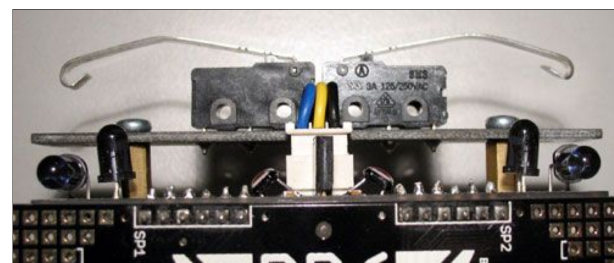
- Die LEDs fangen auch an zu blinken wenn die Akkuspannung zu niedrig ist! Verbinden Sie den Roboter einfach mal mit dem PC und schauen Sie ob beim verbinden eine Warnung wegen zu niedriger Akkuspannung erscheint! Wenn Sie den Reset Taster des Roboters drücken und der Bootloader auf das Startsignal wartet, fangen die vier roten LEDs ebenfalls an zu blinken wenn die Akkuspannung zu niedrig ist! Das Programm lässt sich aber trotzdem noch starten. Allerdings kommt der Roboter in diesem Fall nicht mehr allzuweit und die LEDs sollten auch dann kurz darauf anfangen zu blinken...
- Wenn Sie sich sicher sind, dass die Akkus voll sind und im RP6Loader auch eine Akkuspannung über 6V im Statusfeld angezeigt wird – sollte auch hier wie schon beim vorherigen Punkt auf der Liste eine Fehlermeldung auf der seriellen Schnittstelle ausgegeben werden. Es könnte sein, dass eines oder beide Getriebe blockiert sind. Beispielsweise könnte einer der Stellringe zu fest angezogen sein oder ein Kabel ist zwischen die Zahnräder geraten.
- Im schlimmsten Fall, könnten die Motoren oder gar die Ansteuerungselektronik beschädigt sein. Dann hilft nur ein Austausch der betroffenen Bauteile... Lassen Sie bitte das Selbsttestprogramm 8 laufen (der Roboter darf währenddessen NICHT den Boden berühren – die beiden Raupenketten müssen komplett frei laufen können! Auch nicht mit den Händen blockieren! Sonst schlägt der Test sofort fehl!). Dieses Programm testet die Motoren und die Steuerungselektronik inklusive der Motorstromsensoren. Gibt es Fehlermeldungen? (Am besten den Inhalt des kompletten Terminals über das Menü „Optionen-->Terminal speichern“ in einer Textdatei abspeichern!)

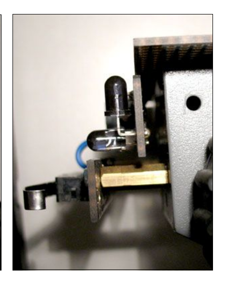
9. Mein Ladegerät lädt die Akkus im Roboter nicht!

- Haben Sie das Ladegerät Polungsrichtig angeschlossen?
- Sind die Akkus richtig eingebaut? Kein Wackelkontakt oder falschrum eingebaut?
- Ist der Stecker des Ladegeräts innen zu groß und hat deshalb keinen Kontakt zu dem inneren Stift der Ladebuchse? (Es gibt verschiedene Versionen dieser Stecker, die kann man normalerweise problemlos austauschen und es sollten verschiedene Varianten bei Ihrem Ladegerät mitgeliefert werden.)

10. Das ACS erkennt Hindernisse nur sehr schlecht!

- Haben Sie es mit verschiedenen Reichweiten/Leistungseinstellungen probiert?
- Sind die IR Leds oder der IR Empfänger verbogen? Die IR LEDs sollten fast gerade von der Sensorplatine abstehen, aber ganz leicht um etwa 5 bis 10° nach aussen gebogen sein. Der IR Empfänger sollte gerade nach oben zeigen (s. Foto im Kapitel 2 und auf dieser Seite).



- Schwarze Objekte erkennt das ACS oftmals nicht besonders gut. Das liegt daran, dass schwarze Oberflächen das IR Licht meist mehr oder weniger gut absorbieren (verschlucken). Das ist also leider normal! Da müssen Sie mit weiteren Sensoren abhelfen (Ultraschall...)!
- Es herrscht starke Fremdlichteinstrahlung in der Umgebung des Roboters, z.B. durch die Sonne, Leuchtstoffröhren von der Raumbeleuchtung oder von der Hintergrundbeleuchtung in Flachbildschirmen und Fernsehern. All diese Lichtquellen können den IR Empfänger stören.

11. Das IRCOMM empfängt keine Signale von meiner Fernbedienung! Der Roboter lässt sich nicht damit steuern!

- Sendet die Fernbedienung wirklich im RC5 Format? Falls Sie das nicht sicher wissen, ist es sehr wahrscheinlich, dass sie das nicht tut. Die Software erkennt nur RC5 Signale. Sie müssen entweder die Fernbedienung auf einen anderen Code umstellen (geht meist nur bei Universalfernbedienungen) oder eine andere Fernbedienung besorgen.
- Die Tastencode Zuordnung ist auf jeder Fernbedienung und bei jedem Hersteller anders. Sie müssen also erstmal die Tastenzuordnung im Programm ändern. Beim RC5 Beispielprogramm steht im Quelltext, wie das geht. Sie müssen sich die Ausgaben im Terminal ansehen! Dort werden die Tastencodes beim RC5 Beispielprogramm ausgegeben!

12. Der Roboter dreht sich nicht exakt um die vorgegebenen Winkel.

- Das ist erstmal normal und es wurde ja oben bereits erläutert warum! Bei Raupenantrieben entstehen immer Abweichungen durch Kettenschlupf. Außerdem müssen die Encoder kalibriert werden bevor das überhaupt richtig klappen kann. Lesen Sie dazu auch Anhang B.

13. Selbst sehr kurze Programme füllen schon 7 KB Programmspeicher! Woran liegt das?

- Wie oben schon erwähnt, wird immer die RP6Library mit eingebunden. Da diese bereits etwas mehr als 6.5KB Programmspeicher belegt, ist das eine normale Programmgröße. Wenn Sie Ihre Programme erweitern, werden Sie feststellen, dass die Programmgröße nicht mehr so stark anwachsen wird. Also keine Sorge, es ist genug Platz im Speicher vorhanden. Naja und falls es doch mal nicht ausreichen sollte, kann man einfach ein Erweiterungsmodul mit zusätzlichem Mikrocontroller verwenden.

14. Meine Programme lassen sich nicht kompilieren – da kommt immer so eine Fehlermeldung!

- Welche Fehlermeldung kommt denn genau? Wenn Sie sich mit so einer Frage an den Support wenden wollen, müssen Sie auch die kompletten Compilerausgaben und Ihren Quelltext mitschicken! Hier schonmal eine Liste mit typischen Fehlern:
  - Sie haben vergessen die RP6Library einzubinden oder die Pfade im Makefile stimmen nicht – wenn Sie ein eigenes Projekt erzeugen müssen Sie evtl. die Pfade im Makefile anpassen! Sonst findet der Compiler die Dateien nicht!
  - Haben Sie irgendwo im Programm ein Semikolon vergessen?
  - Steht irgendwo eine Klammer zuwenig oder zuviel?
  - Haben Sie nicht auf die genaue Schreibweise geachtet? Abgesehen von Kommentaren und den üblichen Formatierungszeichen wie Leerzeichen und Tabulatoren ist jedes Zeichen in C wichtig und es kommt immer auf die korrekte Schreibweise an. WenN hIeR iN dEr Anltng Fheler gemahcT wrden, iss das fAst egaL,, mAn verstht eS TrOtzdEm.! Aber der Compiler versteht dann gar nichts mehr und produziert oft sehr viele Fehlermeldungen obwohl es nur einen einzigen Fehler gibt. Eine automatische Fehlerkorrektur wie wir Menschen sie haben, hat der Compiler leider nicht eingebaut...

15. Meine Programme funktionieren nicht so richtig und der Roboter macht überhaupt nicht was ich will – was mache ich falsch?

- Keine Ahnung ;-) Das müssen Sie schon etwas genauer beschreiben bevor Sie den Support kontaktieren! Vor allem kommen häufig Fragen wie „Warum funktioniert mein Programm nicht?“. Dann sollte aber noch dazu gesagt werden, was das Programm denn überhaupt tun soll! Das wäre sonst nur ein Ratespiel ...

- Häufige Anfängerfehler sind:
  - Ein Semikolon an die falsche Stelle gesetzt – z.B. hinter eine Schleife oder eine if Bedingung. Da kann man zwar oft einfach ein Semikolon hinmachen – aber das ist nicht immer das, was man eigentlich wollte!
  - Klammern bei if-else Konstrukten falsch gesetzt – da kann man sich schonmal vertun wenn man die Blöcke nicht richtig einrückt.
  - Falschen Datentyp für eine Variable verwendet – ein uint8_t kann z.B. nur Werte bis 255 annehmen, aber nicht bis 1500 zählen! Dafür müsste man dann einen uint16_t nehmen! Genausowenig kann ein uint8_t negative Werte enthalten – das geht nur mit Vorzeichenbehafteten Datentypen wie int8_t! Schauen Sie sich dazu nochmal die Tabelle mit den Datentypen ganz am Anfang des C Crahskurses an!
  - Endlosschleife am Ende des Programms vergessen – ohne diese Schleife könnte das Programm unvorhergesehene Dinge anstellen.
  - Sie verwenden den nicht blockierenden Modus der move oder rotate Funktionen, rufen aber nicht ständig die task_motionControl bzw. die task_RP6System Funktion auf! Oder Sie erzeugen längere Pausen mit mSleep in Ihrem Programm. Wenn Sie den nicht blockierenden Modus von den rotate und move Funktionen, oder das ACS o.ä. verwenden, müssen Sie alle Timing Aufgaben die längere Pausen als etwa 10 Millisekunden erfordern, mit den Stopwatches erledigen! mSleep und andere blockierende Funktionen dürfen nicht zusammen mit den nicht blockierenden Funktionen verwendet werden! Lesen Sie am besten das Kapitel über die RP6Library nochmal in Ruhe durch und schauen Sie sich nochmal die Beispielprogramme dazu an!
  - **Immer dran denken die Programme auch abzuspeichern bevor man diese nach einer Änderung im Quelltext erneut kompiliert! Sonst wird noch die alte Version, ohne eben diese Änderung übersetzt! Im Zweifelsfall auch mal MAKE CLEAN ausführen und dann nochmal kompilieren!**

16. Sie haben ein ganz anderes Problem?

- Wichtige Stellen in der Anleitung nochmal lesen hilft oft! Aber natürlich nicht immer...
- Haben Sie schon die neuesten Software Versionen und die neueste Version dieser Anleitung von http://www.arexx.com/ bzw. von der RP6 Homepage http://www.arexx.com/rp6 heruntergeladen?
- Suchfunktion im Forum genutzt? --> http://www.arexx.com/forum/
- Im Roboternetz nachgeschaut? --> http://www.roboternetz.de/
- Haben Sie sich schon allgemein etwas in den beiden Foren umgesehen? (Bitte nicht sofort ein neues Thema posten – erstmal die Suchfunktion verwenden und schauen ob es schon Themen zu Ihrem Problem gibt!)

B – Encoder Kalibrierung

Die Auflösung der Encoder (Drehgeber) ist vom tatsächlichen Durchmesser der Antriebsräder und der Gummiketten abhängig. Die Gummiketten drücken sich in den Untergrund ein und werden auch selbst etwas eingedrückt. Zusätzliche gibt es natürlich Fertigungsschwankungen. Daher kann man den Durchmesser nicht genau berechnen und muss Messreihen durchführen um die tatsächliche Encoderauflösung etwas genauer zu ermitteln.

Die Auflösung ist die Distanz, die der Roboter pro Encoderzählschritt zurücklegt. Theoretisch sind das 0.25mm pro Zählschritt. Aber in der Praxis liegt die tatsächliche Auflösung eher im Bereich von 0.23 bis 0.24mm.

Um die Auflösung zu bestimmen, müssen Sie den Roboter eine möglichst große Distanz geradeaus fahren lassen (z.B. einen Meter oder mehr) und mit einem Maßband genau messen wie weit er tatsächlich gefahren ist. Der Roboter sollte dabei mit dem PC verbunden bleiben und die gezählten Encoderschritte anzeigen. Das USB Kabel und das Flachbandkabel muss locker über dem Roboter hergeführt werden – nicht dran ziehen oder das Kabel zu fest halten! Die vordere Kante der Bumper Platine sollte zu beginn genau auf den Anfang des Maßbandes zeigen. Der Roboter sollte möglichst genau ausgerichtet werden, damit er geradeaus neben dem Maßband herfahren kann.

Man kann (als kleine Übung) z.B. ein Programm schreiben in dem der Roboter genau einen Meter weit fährt. Oder 2m oder eine andere Distanz. Das Programm kann man immer neu kompilieren und wieder in den Roboter laden wenn man die Distanz geändert hat. Das Programm sollte auch in regelmäßigen Intervallen die bereits zurückgelegte Distanz in Encoder Zählschritten ausgeben.

Ein Meter entspricht bei 0.25mm Auflösung genau 4000 Zählschritten. Fährt der Roboter bei dem Test nun aber z.B. nur 96.5cm also 965mm und zählt dabei 4020 Encoderschritte, entspräche das einer Auflösung von etwa 0.24mm. Man teilt hier einfach 965mm durch 4020. Diese Messwerte notieren Sie sich dann in einer Tabelle. Wiederholen Sie den Test noch zwei mal und tragen Sie auch diese Werte in die Tabelle ein. Dann trägt man den Mittelwert daraus in die Datei RP6Lib/RP6Base/RP6Config.h bei `ENCODER_RESOLUTION` ein (relativer Pfad vom Hauptverzeichnis der Beispielprogramme aus gesehen – abspeichern nachher nicht vergessen!), kompiliert das Programm erneut und lädt es in den Roboter. Dann wiederholt man den Test wieder dreimal. Es sollte jetzt schon etwas besser klappen, einen Meter weit zu fahren. Falls nicht, muss man den neu ermittelten Wert wieder in die Datei eintragen. Perfekt bekommt man das allerdings nur schwer hin – jedenfalls nicht ohne zusätzliche Sensoren.

Beim Rotieren auf der Stelle wird die Sache viel komplizierter. Hier rutschen die Ketten während der Drehung nämlich über den Boden und die tatsächlich zurückgelegte Strecke ist kürzer, als die gemessene Strecke. Zudem ist das sehr stark vom Untergrund abhängig, da die Ketten auf Teppichboden „anders rutschen" als z.B. auf Parkett. Von daher muss man immer mit gewissen Ungenauigkeiten von bis zu 10° rechnen. Auch driftet der Roboter auf einigen Untergründen zur Seite weg. Hier muss man ein paar mehr Versuche durchführen.

Wenn man während einer Rotation den Roboter vorne an der Bumperplatine leicht anhebt, so dass er nur noch auf den Hinterrädern dreht, sieht man wie weit er sich eigentlich bei der Drehung bewegen müsste!

Bessere Methoden zur Wegmessung und Positionsbestimmung

Das alles wird nur mit den Encodern nie 100%ig genau werden. Wenn man mit einem Raupenfahrzeug genauer navigieren will, muss man zusätzliche Sensoren verwenden.

Die amerikanische Universität Michigan hat bei einem Ihrer großen Roboter mit Raupenantrieb z.B. einen kleinen Messanhänger gebaut, den der Roboter hinter sich herzieht (Roboter und Anhänger sind dabei über einen schwenkbaren Metallstab verbunden). Dieser Anhänger ermittelt die tatsächliche Position dann ebenfalls über Drehgeber und Absolutwertgeber und ein paar Berechnungen. Nachzulesen z.B. hier:

`http://www-personal.engin.umich.edu/%7Ejohannb/Papers/umbmark.pdf`

ab Seite 20 (bzw. 24 im PDF)
oder hier `http://www.eecs.umich.edu/~johannb/paper53.pdf`

Sowas ähnliches könnte man auch für den RP6 konstruieren (ist allerdings nicht gerade einfach) – evtl. könnte man aber auch mit einem Sensor aus einer optischen Maus experimentieren, den man hinten oder vorne am Roboter anbringt. Zusätzlich oder alternativ dazu noch ein elektronischer Kompass o.ä. den man etwas höher über dem Roboter anbringt (damit er nicht von den Motoren und der restlichen Elektronik gestört wird) und dann sollte es möglich sein, um recht genaue Winkel zu rotieren.

Eine andere Alternative währen Gyro Sensoren (Drehratensensoren) ...

Das ist aber alles noch nicht getestet worden und soll nur ein kleiner Denkanstoß sein, was man noch so alles verbessern und untersuchen könnte... wenn man es nicht so genau braucht – kann man sich natürlich auch um andere Dinge kümmern. Mit einem Maussensor schränkt man schließlich auch die Geländegängigkeit des Roboters ein.

Wirklich genau wird es ohnehin nur, wenn man Landmarken einsetzt. Also Punkte in der Umgebung des Roboters, deren Position genau bekannt sind und die der Roboter irgendwie erkennen kann. Das können z.B. Infrarot Baken sein, die ein Signal aussenden anhand dessen der Roboter die Richtung bestimmen kann, in der diese Baken sich relativ zu seiner aktuellen Position befinden.

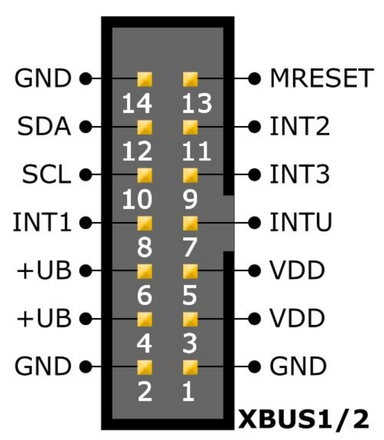
Oder man baut Liniensensoren an den Roboter an, mit denen er Markierungen auf dem Boden erkennen kann.

Es gibt natürlich auch noch viel kompliziertere Methoden dafür... das hier sind nur ein paar einfache Varianten, die ohne große Rechenleistung auskommen. Man könnte z.B. auch eine Kamera an der Decke montieren und den Roboter dann per Funk oder Infrarot von einem PC mit spezieller Bildverarbeitungssoftware aus steuern. Das haben einige Anwender z.B. schon vor einiger Zeit mit dem kleinen ASURO gemacht...

Wenn man den Roboter mit einer TV Fernbedienung fernsteuert, wird man feststellen, dass man die Bewegungen relativ genau steuern kann. Das liegt daran, dass man als Mensch direkt sehen kann, wie und wohin sich der Roboter bewegt! Der Roboter selbst kann das aber leider nicht sehen. Mit einer Kamera an der Decke und z.B. einer deutlich sichtbaren farbigen Markierung oben auf dem Roboter, die auch die Orientierung anzeigt, hätte man allerdings einen externen Sensor, der ähnliches leisten kann...

C – Anschlussbelegungen

In diesem Abschnitt finden Sie die Anschlussbelegungen der wichtigsten Stecker und Lötpads auf dem Mainboard, zusammen mit einigen wichtigen Hinweisen zur Verwendung.



Zunächst der Vollständigkeit halber nochmal die Anschlussbelegung der Erweiterungsstecker mit passendem Textauszug aus Kapitel 2:

Pin 1 liegt auf dem Mainboard immer auf der Seite auf der sich die weisse Beschriftung XBUS1 bzw. XBUS2 befindet bzw. die mit „1" neben dem Stecker beschriftet ist.

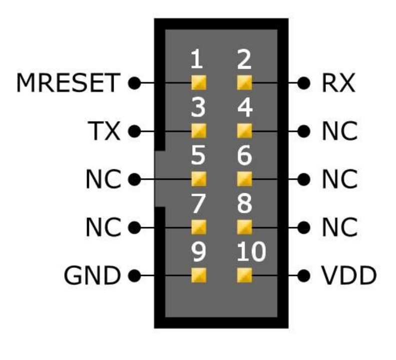
+UB ist die Batteriespannung, VDD die +5V Betriebsspannung, GND bezeichnet den „Minuspol" (GND = Ground, also Masse), MRESET ist das Master Reset Signal, INTx sind die Interruptleitungen, SCL die Takt- und SDA die Datenleitung vom I²C Bus.

Wichtiger Hinweis: **Belasten Sie die Versorgungsleitungen VDD und +UB der Erweiterungsanschlüsse nicht mehr als bis jeweils maximal 1A (gilt für beide Pins ZUSAMMEN! Also jeweils die Pins 4+6 (+UB) und 3+5 (VDD))!**

Alles andere, was Sie evtl. noch brauchen sollten, müssen Sie selbst an die USRBUS Anschlüsse löten. Die USRBUS Anschlüsse sind 1:1 mit den Pads auf der Platine verbunden, d.h. Pin1 vom jew. Stecker ist mit Y1 verbunden, Pin2 mit Y2 etc..

Die Anschlussbelegung der beiden 10 poligen Stecker zur Verbindung mit dem USB Interface haben etwas unterschiedliche Anschlussbelegungen:

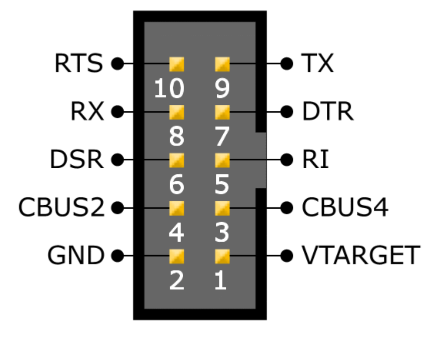
Mainboard:



RX und TX müssen natürlich vertauscht sein, damit die Kommunikation funktionieren kann, und weiterhin ist die Steckerorientierung gespiegelt, damit die Flachbandkabel mit Zugentlastung richtig herum orientiert sind.

(NC heisst „Not Connected" also nicht angeschlossen)

USB Interface:



Beachten Sie auch die in den Grafiken dargestellte Orientierung der Stecker anhand der Pin Nummern und der unterbrochenen schwarzen Linie!

Den UART Anschluss auf dem Mainboard kann man während er nicht mit dem USB Interface verbunden ist auch für andere Zwecke verwenden – z.B. um den Roboter per UART von einem anderen Mikrocontroller zu steuern. Wenn Sie anstelle des USB Interfaces eigene Hardware an das Mainboard anschließen möchten (z.B. ein RS232 Adapter mit MAX232 Pegelwandler oder ein Bluetooth Modul) müssen Sie TX mit RX, RX

mit TX und GND mit GND verbinden. Wenn Sie sich mit der Anschlussbelegung nicht sicher sein sollten, schalten Sie zunächst zum Testen 1k Ohm Widerstände in Serie damit die Treiberstufen des Controllers bei falschem Anschluss nicht überlasten!

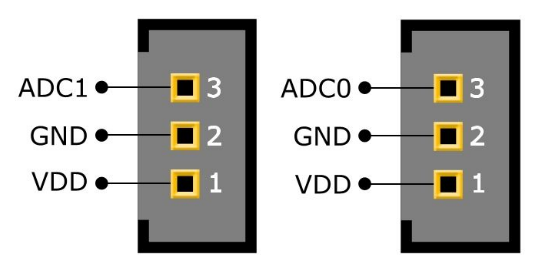
Eine normale RS232 Schnittstelle am PC verwendet 12V Pegel. Der RP6 hat keinen Pegelwandler auf dem Mainboard, da dies mit dem USB Interface nicht notwendig ist. Für eine normale 12V RS232 Schnittstelle muss allerdings ***unbedingt*** ein Pegelwandler wie der MAX232 zwischengeschaltet werden, da der RP6 mit 5V TTL Pegel arbeitet.

Wenn man den Bootloader verwenden möchte, muss MRESET mit RTS verbunden werden (Achtung: Auch RTS hat bei normalen RS232 Schnittstellen natürlich 12V Pegel!). Die anderen Anschlüsse brauchen Sie normalerweise nicht. An VDD liegt die +5V Betriebsspannung vom Roboter an. Kleinere Verbraucher bis etwa 200mA können hier direkt angeschlossen werden. Versuchen Sie nicht den Roboter über diesen Anschluss zu versorgen!

Wenn MRESET auf LOW Pegel geschaltet wird, wird der Roboter zurückgesetzt. So wird die Kommunikation mit dem RP6Loader eingeleitet. Im RP6Loader gibt es ausserdem noch die Optionen den Reset Pin (RTS) zu invertieren („Invert Reset" bzw. „Reset Pin invertieren") und die Übertragungsgeschwindigkeit auf 38.4k Baud zu begrenzen und nicht automatisch beim Program Upload auf 500k Baud umzuschalten („LowSpeed erzwingen"). Die meisten standard RS232 Anschlüsse unterstützen nur maximal 232k Baud.

ADC Anschlüsse

Die Anschlussbelegung der beiden ADC Anschlüsse ist in der nebenstehenden Grafik zu sehen. Sie sind nicht mit Steckern bestückt, daher können Sie beliebige 3 polige Stecker mit Rastermaß 2.54mm verwenden und selbst anlöten. Aber vorsichtig sein und nichts auf dem Mainboard „kaputtlöten"! Das sollte man nur machen wenn man schon Löterfahrung hat. Also im Zweifelsfall besser erstmal mit einem Erweiterungsboard anfangen!



Sie können an die Anschlüsse zwei beliebige analoge Sensoren anschließen (Ausgangsspannung der Sensoren kann im Bereich von 0 bis 5V liegen) und mit 5V Betriebsspannung versorgen. Die ADC Anschlüsse lassen sich natürlich auch als normale I/Os verwenden. Eventuell sollte man noch den großen Elko auf dem Mainboard bestücken – 220 bis 470µF (mehr nicht! Spannungsfestigkeit sollte mindestens 16V betragen) eignet sich gut für die meisten Anwendungen.

Das ist aber nur dann nötig, wenn Sensoren mit hohem Spitzenstrom eingesetzt werden – wie z.B. die beliebten Sharp IR Abstandssensoren. Abblockkondensatoren (100nF) auf dem Mainboard, eignen sich nur für sehr kurze Zuleitungen – bei längeren Leitungen sollten diese direkt an die Sensoren angelötet werden (was aber auch schon bei kurzen Leitungen sehr zu empfehlen ist!).

Alle anderen Anschlüsse sind bereits auf dem Mainboard ausreichend gut beschriftet. Ein Blick in die Schaltpläne, die Sie auf der CD finden können, lohnt sich meistens auch!

D – Entsorgungs- und Sicherheitshinweise

Entsorgungshinweise

Der RP6 darf nicht im normalen Hausmüll entsorgt werden! Er muss wie alle Elektrogeräte beim Wertstoffhof oder an einer anderen Elektro-Altgeräte Sammelstelle abgegeben werden!

Falls Sie Fragen dazu haben, wenden Sie sich an Ihren Händler.

Sicherheitshinweise für Akkus und Batterien

Akkus und Batterien gehören nicht in Kinderhände! Lassen Sie Batterien/Akkus nicht offen herumliegen, es besteht die Gefahr, dass diese von Kindern oder Haustieren verschluckt werden. Suchen Sie im Falle eines Verschluckens sofort einen Arzt auf!

Ausgelaufene oder beschädigte Batterien können bei Berührung mit der Haut Verätzungen verursachen, benutzen Sie deshalb in diesem Fall geeignete Schutzhandschuhe. Achten Sie darauf, dass die Batterien/Akkus nicht kurzgeschlossen oder ins Feuer geworfen werden. Normale Batterien dürfen außerdem nicht aufgeladen werden! Es besteht Explosionsgefahr! Nur darauf ausgelegte, wiederaufladbare Akkumulatoren wie z.B. NiMH Akkus dürfen mit einem passenden Ladegerät geladen werden!

Entsorgungshinweise für Akkus und Batterien

Akkus und Batterien gehören genausowenig in den Hausmüll wie der Roboter selbst! Sie als Endverbraucher sind gesetzlich (Batterieverordnung) zur Rückgabe aller gebrauchten Batterien und Akkus verpflichtet. Eine Entsorgung über den Hausmüll ist untersagt!

Geben Sie defekte/alte Akkus bzw. leere Batterien deshalb nur bei Ihrem Händler, oder einer Batteriesammelstelle Ihrer Gemeinde ab! Sie können gebrauchte Akkus und Batterien auch überall dort abgeben, wo Batterien und Akkus verkauft werden.

Sie erfüllen damit die gesetzlichen Verpflichtungen und leisten Ihren Beitrag zum Umweltschutz!